DIGITAL COMPUTE PLATFORMS (19A02601T)

LECTURE NOTES

III-B.TECH & II - SEM

Prepared by:
Mrs. C. MUNIKANTHA,
Assistant Professor

VEMU INSTITUTE OF TECHNOLOGY
(Approved By AICTE, New Delhi and Affiliated to JNTUA, Ananthapuramu)
Accredited By NAAC, NBA(EEE, ECE & CSE)& ISO: 9001-2015 Certified Institution
Near Pakala, P.Kothakota, Chittoor- Tirupathi Highway
Chittoor, Andhra Pradesh-517 112
Site: www.vemu.org

R19 Regulations

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
(Established by Govt. of A.P., ACT No.30 of 2008)
ANANTHAPURAMU – 515 002 (A.P) INDIA

ELECTICAL AND ELECTORNICS ENGINEERING

DIGITAL COMPUTE PLATFORMS (19A02601T)

Course Objectives:

- Architecture and designing of 8086 Microprocessor with Assembling language programming and interfacing with various modules
- Understand the Interfacing of 8086 with various advanced communication devices
- Designing of 8051 Microcontroller with Assembling language programming and interfacing with various modules To know about Assembly Language Programs for the Digital Signal Processors and usage of Interrupts
- To understand Xilinx programming and understanding of Spartan FPGA board

UNIT-I: INTRODUCTION TO MICROPROCESSORS

Historical background- Evolution of microprocessors up to 64-bit. Architecture of 8086 microprocessor, special function of general purpose registers. 8086 flag registers and functions of 8086 flags – Addressing modes of 8086 – Instruction set of 8086 – Assembler directives - Pin diagram 8086 – Minimum mode and maximum mode of operation - Timing diagrams - CISC and ARM Processors.

Learning Outcomes:-

After completion of this unit student will

- To know about 8086 as one of digital compute platforms
- To know about Architecture and functions of 8086
- To understand about instruction set
- To know about pin and timing diagrams
- To know about processors CISC and ARM

UNIT II: ASSEMBLY LANGUAGE PROGRAMMING & I/O INTERFACE

Assembler directives – macros – simple programs involving logical – branch instructions – sorting – evaluating arithmetic expressions - string manipulations – 8255 PPI - various modes of operation - A/D - D/A converter interfacing, Memory interfacing to 8086 – interrupt structure of 8086 – vector interrupt table – interrupt service routine – interfacing interrupt controller 8259 - Need of DMA – serial communication standards – serial data transfer schemes.

Learning Outcomes:-

After completion of this unit student will

- To understand the programming features of assembly language as one of digital compute platforms
- To know about evaluation of expressions, strings

- To understand about interfacing with A/D-D/A converters
- To understand about interrupt structures and various service routines in 8086
- To know about data transfer scheme

UNIT III:

8051 MICRO CONTROLLER PROGRAMMING AND APPLICATIONS

Introduction to micro controllers, Functional block diagram, Instruction sets and addressing modes, interrupt structure – Timer – I/O ports – serial communication. Data transfer, manipulation, Control and I/O instructions – simple programming exercises key board and display interface – Closed loop control of servo motor – stepper motor control.

Learning Outcomes:-

After completion of this unit student will

- To understand about 8051 Microcontroller as one of the digital compute platforms
- To know about instruction sets of 8051
- To know about data transfer manipulations
- To understand and write programming using 8051
- To know about a few applications of 8051 like servo motor, stepper motor

UNIT IV:Introduction to the TMS320LF2407 DSP Controller

Introduction to the TMS320LF2407 DSP Controller Basic architectural features - Physical Memory - Software Tools. Introduction to Interrupts - Interrupt Hierarchy - Interrupt Control Registers. C2xx DSP CPU and Instruction Set: Introduction & code Generation - Components of the C2xx DSP core - Mapping External Devices to the C2xx core - peripheral interface - system configuration registers - Memory - Memory Addressing Modes - Assembly Programming Using the C2xx DSP Instruction set.

Learning Outcomes:-

After completion of this unit student will

- To know about features of DSP controller C2xx as one of the DCPs
- To know about various instruction sets, control registers of C2xx DSP core
- To know about mapping of external devices to the DSP core
- To know about assembly programming using the instruction sets of TMS320LF2407 DSP controller

UNIT V: FPGA Introduction to Field Programmable Gate Arrays

CPLD Vs FPG Types of FPGA– Xilinx, XC3000 series - Configurable logic Blocks (CLB) – Input / Output Block (IOB) – Programmable Interconnect Point (PIP) – Xilinx 4000 series – HDL programming –overview of Spartan 3E and Virtex II pro FPGA boards- case study.

Learning Outcomes:-

After completion of this unit student will

- To know about FPGA as one of the digital compute platforms.
- To know about various types of FPGA
- To know about programmable inter connect points
- To understand about Xilinx-HDL programming
- To know about applications of FPGA with a case study.

Course Outcomes:

1. Understand the basic architecture & pin diagram of 8086 microprocessor.
2. Assembly language programming to perform a given task, Interrupt service routines for all interrupt types
3. Microprocessor and Microcontroller designing for various applications.
4. Write Assembly Language Programs for the Digital Signal Processors and use Interrupts for real-time control applications
5. Write Xilinx programming and understanding of Spartan FPGA board

TEXT BOOKS

1.Ramesh S. Gaonkar, "Microprocessor Architecture Programming and applications with 8085", Penram Intl. Publishing, 6th Edition, 2013

2. Ray A. K., Bhurchandi K. M., "Advanced Microprocessor and Peripherals", Tata McGraw- Hill Publications, 3rd Edition, 2013.

REFERENCE BOOKS 1. Douglas V Hall, "Microprocessor and Interfacing", 2 nd Edition, Tata McGraw hill, 1992 2. Nilesh B Bahadure, "Microprocessor", PHI, 2010.

3. Kenneth J Ayala, "The 8051 Micro Controller Architecture, Programming and Applications Pearson International publishing (India).

4. Hamid A. Tolyat, "DSP Based Electro Mechanical Motion Control", CRC press, 2004.

5. Application Notes from the webpage of Texas Instruments.

6. XC 3000 series datasheets (version 3.1). Xilinx Inc., USA, 1998

7. XC 4000 series datasheets (version 1.6). Xilinx Inc., USA, 1999 8. Wayne Wolf, FPGA basedsystem design, Prentice hall, 2004.

UNIT-1

C

INTRODUCTION:

Microprocessor acts as a CPU in a microcomputer. It is present as a single ICchip in a icrocomputer.

Microprocessor is the heart of the machine.

A Microprocessor is a device, which is capable of

1. Receiving Input
2. Performing Computations
3. Storing data and instructions
4. Display the results
5. Controlling all the devices that perform the above 4 functions.

The device that performs tasks is called Arithmetic Logic Unit (ALU). A single chip called Microprocessor performs these tasks together with other tasks.

"A MICROPROCESSOR is a multipurpose programmable logic device that reads binary instructions from a storage device called memory accepts binary data as input and processes data according to those instructions and provides results as output ."

EVOLUTION OF MICROPROCESSORS:

The microprocessor age began with the advancement in the IC technology to put all necessary functions of a CPU into a single chip.

Intel started marketing its first microprocessor in the name of Intel 4004 in 1971. This was a4-bit microprocessor having 16-pins in a single chip of PMOS technology. This was called the first generation microprocessor. The Intel 4004 along with few other devices was used for making calculators. The 4004 instruction set contained only 45 instructions. Later in 1971, INTEL Corporation released the 8008 an extended 8-bit version of the 4004 microprocessor. The 8008 addressed an expanded memory size (16KB) and 48 instructions.

Limitations of first generation microprocessors is small memory size, slow speed and instruction set limited its usefulness.

Second generation microprocessors:

The second generation microprocessor using NMOS technology appeared in the market in the year 1973. The Intel 8080, an 8-bit microprocessor, of NMOS technology was developed in the year 1974 which required only two additional devices to design a functional CPU. The advantages of second generation microprocessors were

- Large chip size (170x200 mil) with 40-pins.
- Ability to address large memory space (64-K Byte) and I/O ports(256).
- More chips on decoding circuits.
- More powerful instruction sets.
- Dissipate less power.
- Better interrupt handling facilities.
- Cycle time reduced to half (1.3 to 9 m sec.)
- Sized 70x200 mil) with 40-pins.
- Less Support Chips Required
- Used Single Power Supply
- Faster Operation

The 8080 microprocessor addresses more memory and execute additional instructions, but executes them 10 times faster than 8008.The 8080 has memory of 64 KB whereas for 8008 16 KB only. In 1977, INTEL, introduced 8085 which was an updated version of 8080 last 8-bit processor.

The main advantages of 8085 were its internal clock generator, internal system controller and higher clock frequency.

Third Generation Microprocessor:

In 1978, INTEL released the 8086 microprocessor, a year later it released 8088. Both devices were 16 bit microprocessors, which executed instructions in less than 400ns.The 8086 and 8088 addresses 1MB of memory and rich instruction set to 246.16-bit processors were designed using HMOS technology. The Intel 80186 and 80188 were the improved versions of Intel 8086 and8088, respectively. In addition to 16-bit CPU, the 80186 and 80188 had programmable peripheral devices integrated on the same package.

Fourth Generation Microprocessor:

The single chip 32-bit microprocessor was introduced in the year 1981 by Intel as iAPX 432. The other 4^{th}generation microprocessors were; Bell Single Chip Bellmac-32, Hewlett-Packard, National NSl 6032,Texas Instrument99000. Motorola 68020 and 68030. The Intel in the year 1985 announced the 32-bit microprocessor(80386). The 80486 has already been announced and is also a 32-bit microprocessor.

The 80486 is a combination 386 processor a math coprocessor, and a cache memory controller on a single chip.

The Pentium is a 64-bit superscalar processor. It can execute more than one instruction at a time and has a full 64-bit data bus and 32-bit address bus. Its performance is double than 80486.

Features of 8086:

•It is a 16-bit μp.

•8086 has a 20 bit address bus can access up to 2^20 memory locations (1 MB).

•It can support up to 64K I/O ports. •It

It provides 14, 16 -bit registers.

•It has multiplexed address and

data bus AD0- AD15 and A16 A19. –

•It requires single phase clock with 33% duty cycle to provide internal timing.

•8086 is designed to operate in two modes, Minimum and Maximum.

•It can pre-fetches up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.

•It requires +5V power supply. •A
40 pin dual in line package.

Architecture of 8086:

- 8086 has two blocks BIU and EU.

  The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.

  EU executes instructions from the instruction byte queue.

  Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.

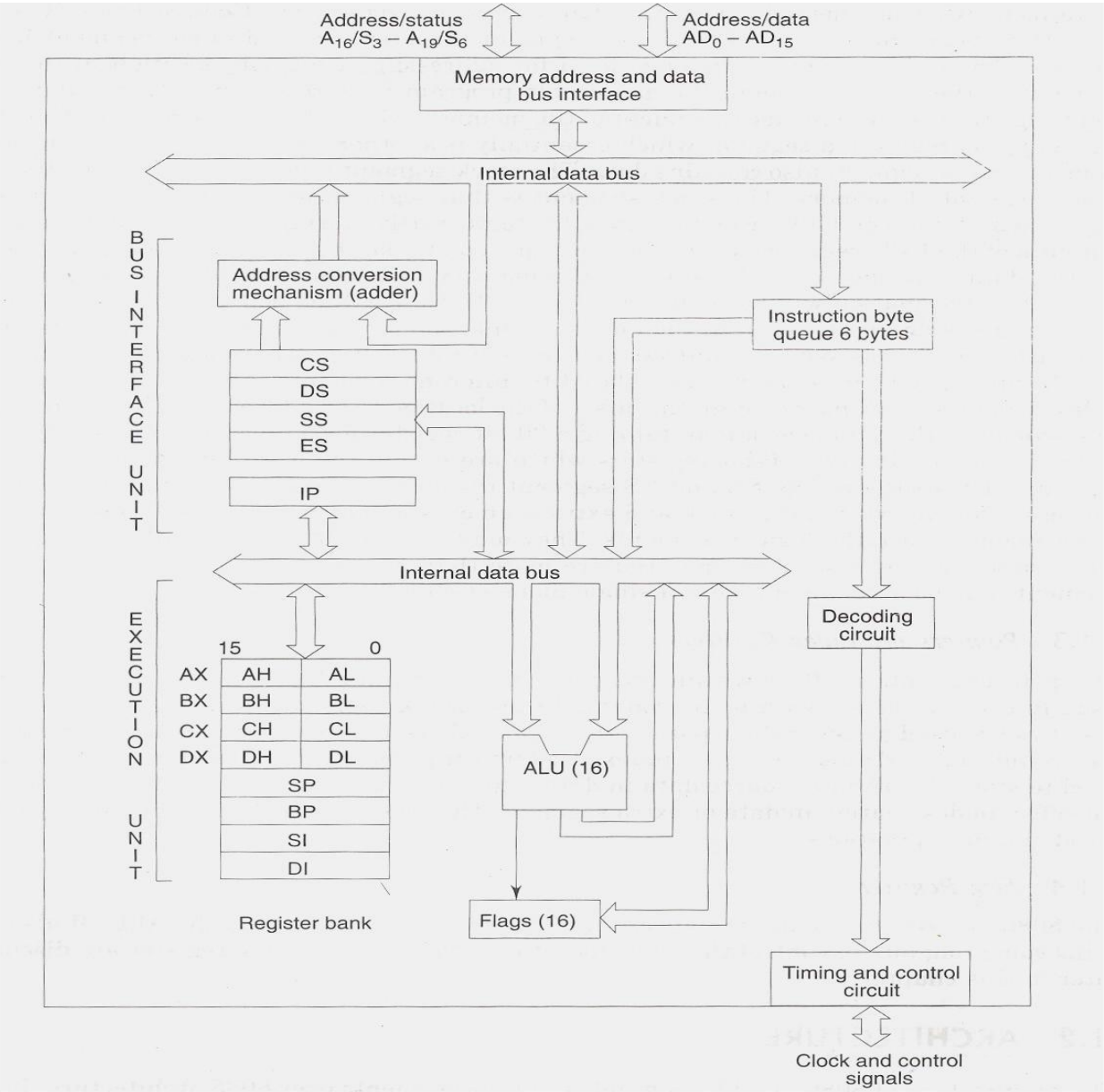
  BIU contains Instruction queue, Segment registers, IP, address adder.

EU contains control circuitry, Instruction decoder, ALU, Flag register.

The BIU is responsible for performing all external bus operations.
Specifically it has the following functions:

Instructions fetch Instruction queuing, Operand fetch and storage, Address relocation and Bus control.

The BIU uses a mechanism known as an instruction stream queue to implement pipeline architecture.



- Bus Interface Unit:

  It provides full 16 bit bidirectional data bus and
  20 bit address bus.

This queue permits pre-fetch of up to six bytes of instruction code. Whenever the queue of the BIU

is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by pre-fetching the next sequential instruction.

These pre-fetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.

- After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.
- The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
- These intervals of no bus activity, which may occur between bus cycles, are known as idle state.
- If the bus is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
- The BIU also contains a dedicated adder which is used to generate the 20 bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
- For example: The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
- The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

Execution Unit:

- The EU extracts instructions from top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bus cycles to memory or I/O and perform the operation specified by the instruction on the operands.
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
- If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
- When ever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

Register organization of 8086:

The 8086 has four groups of the user accessible internal registers. They are the instruction pointer, four data registers, four pointer and index register, four segment registers. The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the status register, with 9 of bits implemented for status and control flags.

There are four different 64 KB segments for instructions, stack, data and extra data. To Specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By

default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

•Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment.DS register can be changed directly using POP and LDS instructions.

•Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

•Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

•Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation,.

•Data register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

•The following registers are both general and index registers:

•Stack Pointer (SP) is a 16-bit register pointing to program stack.

•Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

•Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

•Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

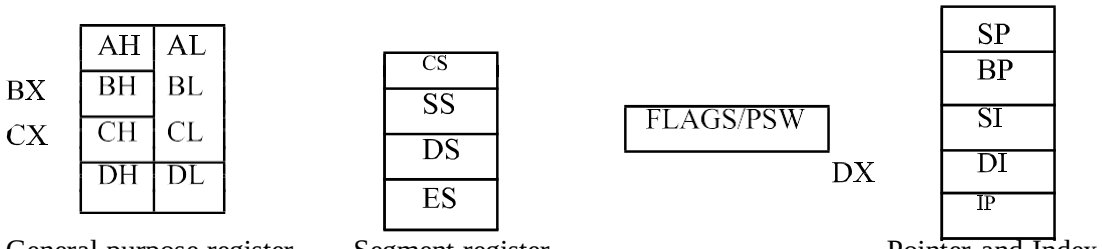
Instruction Pointer (IP) register acts as a program counter for 8086. It points to the address of the next instruction to be executed Its content is automatically incremented when the program execution of a

program proceeds further. The contents of IP and CS register are used to compute the memory address of the instruction code to be fetched.

(

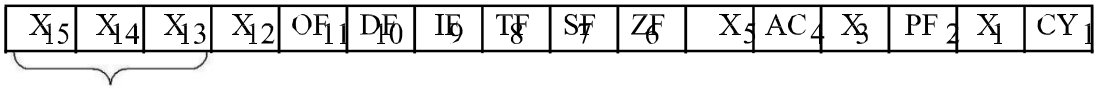
AX

General data registers:



General purpose register Segment register Pointer and Index

Flag register of 8086: It is a 16-bit register, also called flag register or Program Status Word (PSW). Seven bits remain unused while the rest nine are used to indicate the conditions of flags. The status flags of the register are shown below in Fig.



- Out of nine flags, six are condition flags and three are control flags. The control flags
- are TF (Trap), IF (Interrupt) and DF (Direction) flags, which can be set/reset by the
- programmer, while the condition flags [OF (Overflow), SF (Sign), ZF (Zero), AF (Auxiliary
- Carry), PF (Parity) and CF (Carry)] are set/reset depending on the results of some arithmetic or logical operations during program execution.
- CF is set if there is a carry out of the MSB position resulting from an addition operation or if a borrow is needed out of the MSB position during subtraction.
- PF is set if the lower 8-bits of the result of an operation contains an even number of 1's. AF is set if there is a carry out of bit 3 resulting from an addition operation or borrow required from bit 4 into bit 3 during subtraction operation.
- ZF is set if the result of an arithmetic or logical operation is zero.
- SF is set if the MSB of the result of an operation is 1. SF is used with unsigned numbers.
- OF is used only for signed arithmetic operation and is set if the result is too large to be fitted in the number of bits available to accommodate it.

  The three control flags of 8086 are TF, IF and DF. These three flags are programmable, i.e., can be set/reset by the programmer so as to control the operation of the processor.

- When TF (trap flag) is set (=1), the processor operates in single stepping mode–i.e., pausing after each instruction is executed. This mode is very useful during program development or program debugging.
- When an interrupt is recognized, TF flag is cleared. When the CPU returns to the main program from ISS (interrupt service subroutine), by execution of IRET in the last line of ISS, TF flag is restored to its value that it had before interruption.

TF cannot be directly set or reset. So indirectly it is done by pushing the flag register on the stack, changing TF as desired and then popping the flag register from the stack.

When IF (interrupt flag) is set, the maskable interrupt INTR is enabled otherwise disabled (i.e., when IF = 0).

IF can be set by executing STI instruction and cleared by CLI instruction. Like TF flag, when an interrupt is recognized, IF flag is cleared, so that INTR is disabled. In the last line of ISS when IRET is encountered, IF is restored to its original value. When 8086 is reset, IF is cleared, i.e., resetted.

DF (direction flag) is used in string (also known as block move) operations. It can be set by STD instruction and cleared by CLD. If DF is set to 1 and MOVS instruction is executed, the contents of the index registers DI and SI are automatically decremented to access the string from the highest memory location down to the lowest memory location.

ADDRESSING MODES OF 8086:

Addressing modes indicates way of locating data or operands. Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes. Thus the addressing modes describe the types of operands and the way they are accessed for executing an instruction.

According to the flow of instruction execution, the instruction may be categorized as:
Sequential Control flow instructions Control Transfer instructions

- Sequential Control flow instructions: In this type of instruction after execution control can be transferred to the next immediately appearing instruction in the program.
  The addressing modes for sequential control transfer instructions are as follows:
- Immediate addressing mode: In this mode, immediate is a part of instruction and appears in the form of successive byte or bytes.
  Example: MOV CX, 0007H; Here 0007 is the immediate data

- Direct Addressing mode: In this mode, the instruction operand specifies the memory address where data is located.
  Example: MOV AX, [5000H]; Data is available in 5000H memory location
  Effective Address (EA) is computed using 5000H as offset address and content of DS as segment address.

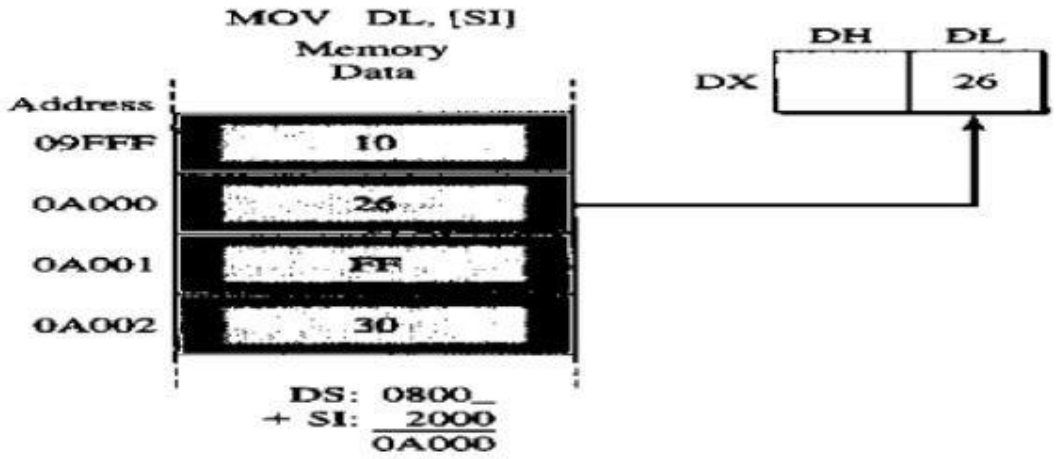
EA=10H*DS+5000H

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

Example: MOV AX, BX;

- Register Indirect addressing mode: In this mode, instruction specifies a register containing an address, where data is located. This addressing mode works with SI, DI, BX and BP registers.
  Example: MOV AX, [BX];
  EA=10H * DS +[BX]
- Indexed Addressing mode: 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides. DS and ES are default segments for index registers SI and DI.
  DS=0800H, SI=2000H,
  MOV DL, [SI]
  Example: MOV AX, [SI];
  EA=10H * DS + [SI]

- Register Relative Addressing mode: In this mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI, DI in the default segments.
  Example: MOV AX, 50H [BX];
  EA=10H * DS + 50H +[BX]



Based Indexed Addressing mode: In this mode, the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

Example: MOV AX, [BX] [SI];

EA=10H * DS + [BX] + [SI]

- Relative Based Indexed Addressing mode: In this mode, 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.
  Example: MOV AX, 50H [BX] [SI];
  EA=10H * DS + 50H + [BX] +[SI]

- Control Transfer Instructions: In control transfer instruction, the control can be transferred to

some predefined address or the address somehow specified in the instruction after their execution.

For the control transfer instructions, the addressing modes depend upon whether the destination

location is within the segment or different segments. It also depends upon the method of passing the destination address to the processor. Depending on this control transfer instructions are categorized as follows:

- Intra segment Direct mode: In this mode, the address to which control is to be transferred lies in the same segment in which control transfer instruction lies and appears directly in the instruction as an immediate displacement value.
- Intra segment Indirect mode: In this mode, the address to which control is to be transferred lies in the same segment in which control transfer instruction lies but it is passed to the instruction indirectly.
- Inter segment Direct mode: In this mode, the address to which control is to be transferred lies in a different segment in which control transfer instruction lies and appears directly in the instruction as an immediate displacement value.
- Inter segment Indirect mode: In this mode, the address to which control is to be transferred lies in a different segment in which control transfer instruction lies but it is passed to the instruction indirectly.

Memory Segmentation for 8086:

8086, via its 20-bit address bus, can address 220 = 1,048,576 or 1 MB of different memory locations. Thus the memory space of 8086 can be thought of as consisting of 1,048,576 bytes or 524,288 words. The memory map of 8086 is shown in Figure where the whole memory space starting from 00000 H to FFFFF H is divided into 16 blocks—each one consisting of 64KB.

1 MB memory of 8086 is partitioned into 16 segments—each segment is of 64 KB length. Out of these 16 segments, only 4 segments can be active at any given instant of time- these are code segment, stack segment, data segment and extra segment. The four memory segments that the CPU works with at any time are called currently active segments. Corresponding to these four segments, the registers used are Code Segment Register (CS), Data Segment Register (DS), Stack Segment Register (SS) and Extra Segment Register (ES) respectively. Each of these four registers is 16-bits wide and user accessible–i.e., their content scan be changed by software.

The code segment contains the instruction codes of a program, while data, variables and constants are held in data segment. The stack segment is used to store interrupt and subroutine return addresses. The extra segment contains the destination of data for certain string instructions. Thus 64 KB are available for program storage (in CS) as well as for stack (in SS) while128 KB of space can be utilized for data storage (in DS and ES).One restriction on the base address (starting address) of a segment is that it must reside on a 16-byte address memory—examples being 00000 H, 00010 H or 00020 H, etc.

Memory segmentation, as implemented for 8086, gives rise to the following advantages:

- Although the address bus is 20-bits in width, memory segmentation allows one to work with registers having width 16-bits only.
- It allows instruction code, data, stack and portion of program to be more than 64 KB long by using more than one code, data, extra segment and stack segment.
- In a time-shared multitasking environment when the program moves over from one user's program to another, the CPU will simply have to reload the four segment registers with the segment starting addresses assigned to the current user's program.

User's program (code) and data can be stored separately.

- Because the logical address range is from 0000 H to FFFF H, the same can be loaded at any place in the memory.

Instruction Set of 8086:

There are 117 basic instructions in the instruction set of 8086.The instruction set of 8086 can be divided into the following number of groups, namely:

1. Data copy / Transfer instructions
2. Arithmetic and Logical instructions
3. Branch instructions
4. Loop instructions
5. Machine control instructions
6. Flag Manipulation instructions
7. Shift and Rotate instructions
8. String instructions

Data copy / Transfer instructions: The data movement instructions copy values from one location to another. These instructions include MOV, XCHG, LDS, LEA, LES, PUSH, PUSHF, PUSHFD, POP,POPF, LAHF, AND SAHF.

MOV The MOV instruction copies a word or a byte of data from source to a destination. The destination can be a register or a memory location. The source can be a register, or memory location or

immediate data. MOV instruction does not affect any flags. The mov instruction takes several different forms:

The MOV instruction cannot:

1. Set the value of the CS and IP registers.
2. Copy value of one segment register to another segment register (should copy to general register first). MOV CS, DS (Invalid)
3. Copy immediate value to segment register (should copy to general register first). MOV CS, 2000H (Invalid)

Example:

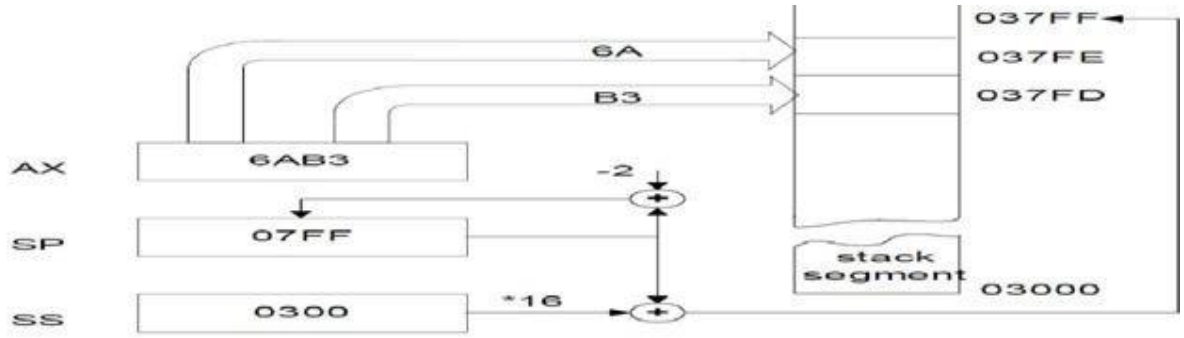
```
ORG 100h
MOV AX, 0B800h ;        set AX = B800h
MOV DS, AX ;            copy value of AX to DS.
MOV CL, 'A' ;       CL = 41h (ASCII code).
```

The XCHG Instruction: Exchange This instruction exchanges the contents of the specified source and destination operands, which may be registers or one of them, may be a memory location. However, exchange of data contents of two memory locations is not permitted.

```
Example: MOV AL, 5 ; AL = 5
         MOV BL, 2 ; BL = 2
         XCHG AL,BL ; AL = 2, BL = 5
```

PUSH: Push to stack; this instruction pushes the contents of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after each execution of the instruction. The actual current stack-top is always occupied by the previously pushed data. Hence, the push operation decrements SP by two and then stores the two byte contents of the operand onto the stack. The higher byte is pushed first and then the lower byte. Thus out of the two decremented stack addresses the higher byte occupies the higher address and the lower byte occupies the lower address.

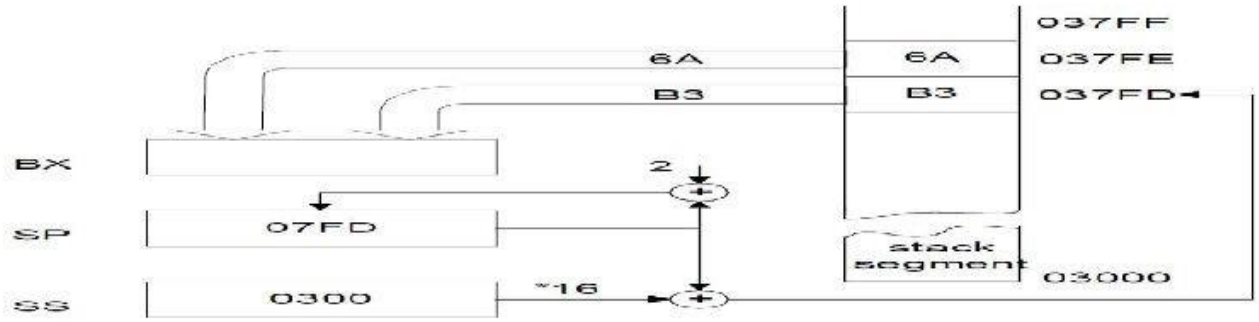
1. PUSH AX
2. PUSH DS
3. PUSH [500OH] ; Content of location 5000H and 5001 H in DS are pushed onto the stack.



The effect of PUSH AX instruction

POP: Pop from Stack this instruction when executed loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer as usual. The stack pointer is incremented by 2. The POP instruction serves exactly opposite to the PUSH instruction.

1. POP BX



The effect of POP BX instruction

2. POP DS
3. POP [5000H]

PUSHF: Push Flags to Stack The push flag instruction pushes the flag register on to the stack; first the upper byte and then the lower byte will be pushed on to the stack. The SP is decremented by 2, for each push operation. The general operation of this instruction is similar to the PUSH operation.

POPF: Pop Flags from Stack The pop flags instruction loads the flag register completely (both bytes) from the word contents of the memory location currently addressed by SP and SS. The SP is incremented by 2for each pop operation.

LAHF: Load AH from Lower Byte of Flag This instruction loads the AH register with the lower byte of the flag register. This instruction may be used to observe the status of all the condition code flags (except overflow) at a time.

SAHF: Store AH to Lower Byte of Flag Register This instruction sets or resets the condition code flags (except overflow) in the lower byte of the flag register depending upon the corresponding bit positions in AH. If a bit in AH is 1, the flag corresponding to the bit position is set, else it is reset.

LEA: Load Effective Address The load effective address instruction loads the offset of an operand in the specified register. This instruction is similar to MOV, MOV is faster than LEA.

LEA cx, [bx+si] ; CX (BX+SI) mod 64K If bx=2f00 H; si=10d0H cx 3fd0H

The LDS AND LES instructions:

• LDS and LES load a 16-bit register with offset address retrieved from a memory location then load either DS or ES with a segment address retrieved from memory.

This instruction transfers the 32-bit number, addressed by DI in the data segment, into the BX and DS registers.

• LDS and LES instructions obtain a new far address from memory.

–offset address appears first, followed by the segment address

• This format is used for storing all 32-bit memory addresses.

• A far address can be stored in memory by the assembler.

LDS BX,DWORD PTR[SI]
BL [SI];
BH [SI+1]
DS [SI+3:SI+2]; in the data segment

LES BX,DWORD PTR[SI]
BL [SI];
BH [SI+1]
ES [SI+3:SI+2]; in the extra segment

I/O Instructions: The 80x86 supports two I/O instructions: in and out15. They take the forms: In ax, port

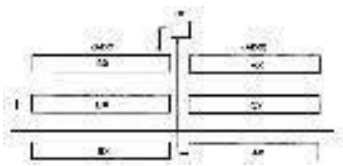
in ax, dx
out port, ax
out dx, ax
port is a value between 0 and 255.

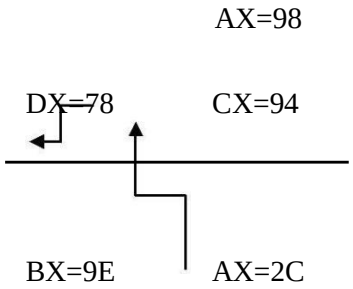
The in instruction reads the data at the specified I/O port and copies it into the accumulator. The out instruction writes the value in the accumulator to the specified I/O port.

Arithmetic instructions: These instructions usually perform the arithmetic operations, like addition, subtraction, multiplication and division along with the respective ASCII and decimal adjust instructions. The increment and decrement operations also belong to this type of instructions.

The ADD and ADC instructions: The add instruction adds the contents of the source operand to the destination operand. For example, add ax, bx adds bx to ax leaving the sum in the ax register. Add computes dest :=dest+source while adc computes dest :=dest+source+C where C represents the value in the carry flag. Therefore, if the carry flag is clear before execution, adc behaves exactly like the add instruction.

Example: CF=1





Both instructions affect the flags identically. They set the flags as follows:

• The overflow flag denotes a signed arithmetic overflow.

• The carry flag denotes an unsigned arithmetic overflow.

• The sign flag denotes a negative result (i.e., the H.O. bit of the result is one).

• The zero flag is set if the result of the addition is zero.

• The auxiliary carry flag contains one if a BCD overflow out of the L.O. nibble occurs.

• The parity flag is set or cleared depending on the parity of the L.O. eight bits of the result. If there is even number of one bits in the result, the ADD instructions will set the parity flag to one (to denote even parity). If there is an odd number of one bits in the result, the ADD instructions clear the parity flag (to denote odd parity).

The INC instruction: The increment instruction adds one to its operand. Except for carry flag, inc sets the flags the same way as Add ax, 1 same as inc ax. The inc operand may be an eight bit, sixteen bit. The inc instruction is more compact and often faster than the comparable add reg, 1 or add mem, 1 instruction.

The AAA and DAA Instructions

The aaa (ASCII adjust after addition) and daa (decimal adjust for addition) instructions support BCD arithmetic. BCD values are decimal integer coded in binary form with one decimal digit(0..9) per nibble. ASCII (numeric) values contain a single decimal digit per byte, the H.O. nibble of the byte should contain zero (30 ….39).

The aaa and daa instructions modify the result of a binary addition to correct it for ASCII or decimal arithmetic. For example, to add two BCD values, you would add them as though they were binary numbers and then execute the daa instruction afterwards to correct the results.

Note: These two instructions assume that the add operands were proper decimal or ASCII values. If you add binary(non-decimal or non-ASCII) values together and try to adjust them with these instructions, you will not produce correct results.

Aaa (which you generally execute after an add, adc, or xadd instruction) checks the value in al for BCD overflow. It works according to the following basic algorithm:

if ((al and 0Fh) > 9 or (AuxC =1)) then al := al + 6

else

ax := ax +6 endif

ah := ah + 1

AuxC := 1 ;Set auxilliary carry Carry := 1 ; and carry flags. Else

AuxC := 0 ;Clear auxilliary carry Carry := 0 ; and carry flags.

add al=08 +06; al=0E >9 al=0E+06=04
ah=00+01=01
al=04+03=08, now al<9,
so only clear ah=0
endif
al := al and 0Fh

The aaa instruction is mainly useful for adding strings of digits where there is exactly one decimal digit per byte in a string of numbers.

The daa instruction functions like aaa except it handles packed BCD values rather than the one digit per byte unpacked values aaa handles. As for aaa, daa's main purpose is to add strings of BCD digits (with two digits per byte). The algorithm for daa is

if ((AL and 0Fh) > 9 or (AuxC = 1)) then
al := al + 6
AuxC := 1 ;Set Auxilliary carry.
Endif
if ((al > 9Fh) or (Carry = 1)) then
al := al + 60h
Carry := 1; ;Set carry flag.
Endif

al=24+77=9B, as B>9 add 6 to al
al=9B+06=A1, as higher nibble A>9, add 60 to al, al=A1+60=101
Note: if higher or lower nibble of AL <9 then no need to add 6 to AL

EXAMPLE:

Assume AL = 0 0 1 1 0 1 0 1, ASCII 5 BL = 0 0 1 1 1 0 0 1, ASCII 9

ADDAL,BL Result: AL= 0 1 1 0 1 1 1 0 = 6EH,which is incorrect
BCD AAA Now AL = 00000100, unpacked BCD 4.
CF = 1 indicates answer is 14 decimal

NOTE: OR AL with 30H to get 34H, the ASCII code for 4. The AAA instruction works only on the AL register. The AAA instruction updates AF and CF, but OF, PF, SF, and ZF are left undefined.

EXAMPLES:

AL = 0101 1001 = 59 BCD ; BL = 0011 0101 = 35
BCD ADD AL, BL AL = 1000 1110 = 8EH

DAA Add 01 10 because 1110 > 9 AL = 1001 0100 = 94
BCD AL = 1000 1000 = 88 BCD BL = 0100 1001 = 49 BCD
ADD AL, BL AL = 1101 0001, AF=1

DAA Add 0110 because AF =1, AL = 11101 0111 =
D7H 1101 > 9 so add 0110 0000
AL = 0011 0111= 37 BCD, CF =1
The DAA instruction updates AF, CF, PF, and ZF. OF is undefined after a DAA instruction.

The SUBTRACTION instructions: SUB, SBB, DEC, AAS, and DAS

The sub instruction computes the value dest :=dest - src. The sbb instruction computes dest :=dest src - C.

The sub, sbb, and dec instructions affect the flags as follows:

• They set the zero flag if the result is zero. This occurs only if the operands are equal for sub and sbb. The dec instruction sets the zero flag only when it decrements the value one.

• These instructions set the sign flag if the result is negative.

• These instructions set the overflow flag if signed overflow/underflow occurs.

• They set the auxiliary carry flag as necessary for BCD/ASCII arithmetic.

• They set the parity flag according to the number of one bits appearing in the result value.

• The sub and sbb instructions set the carry flag if an unsigned overflow occurs. Note that the dec instruction does not affect the carry flag.

The aas instruction, like its aaa counterpart, lets you operate on strings of ASCII numbers with one decimal digit (in the range 0..9) per byte. This instruction uses the following algorithm:

if ((al and 0Fh) > 9 or AuxC = 1)
then al := al - 6

ah := ah - 1

AuxC := 1 ;Set auxilliary
carry Carry := 1 ; and carry
flags. else

AuxC := 0 ;Clear Auxilliary
carry Carry := 0 ; and carry flags.
endif

al := al and 0Fh

The das instruction handles the same operation for BCD values, it uses the following algorithm:

if ((al and 0Fh) > 9 or (AuxC = 1))
then al := al -6

AuxC =
1 endif

if (al > 9Fh or Carry = 1)
then al := al - 60h

Carry := 1 ;Set the Carry
flag. Endif

EXAMPLE:

ASCII 9-ASCII 5 (9-5)

AL = 00111001 = 39H = ASCII 9
BL = 001 10101 = 35H = ASCII 5

SUB AL, BL Result: AL = 00000100 = BCD 04 and CF =
0 AAS Result: AL = 00000100 = BCD 04 and CF = 0

no borrow required
ASCII 5-ASCII 9 (5-9)

Assume AL = 00110101 = 35H ASCII
5 and BL = 0011 1001 = 39H = ASCII 9

SUB AL, BL Result: AL = 11111100 = - 4 in 2s complement and CF
=1 AAS Result: AL = 00000100 = BCD 04 and CF = 1, borrow needed

EXAMPLES:

AL 1000 0110 86 BCD ; BH 0101 0111 57 BCD

SUB AL,BH AL 0010 1111 2FH, CF = 0

DAS Lower nibble of result is 1111, so DAS automatically

subtracts 0000 0110 to give AL = 00101001 29 BCD
AL 0100 1001 49 BCD BH 0111 0010 72 BCD SUB
AL,BH AL 1101 0111 D7H, CF = 1

DAS Subtracts 0110 0000 (- 60H) because 1101 in upper nibble > 9
AL = 01110111= 77 BCD, CF=1 CF=1 means borrow was needed

The CMP Instruction: The cmp (compare) instruction is identical to the sub instruction with one crucial difference–it does not store the difference back into the destination operand. The syntax for the cmp instruction is very similar to sub, the generic form is cmpdest, src

Consider the following cmp instruction: cmp ax, bx

This instruction performs the computation ax-bx and sets the flags depending upon the result of the computation. The flags are set as follows:

Z: The zero flag is set if and only if ax = bx. This is the only time ax-bx produces a zero result. Hence, you can use the zero flag to test for equality or inequality.

S: The sign flag is set to one if the result is negative.

O: The overflow flag is set after a cmp operation if the difference of ax and bx produced an overflow or underflow.

C: The carry flag is set after a cmp operation if subtracting bx from ax requires a borrow. This occurs only when ax is less than bx where ax and bx are both unsigned values.

The Multiplication Instructions: MUL, IMUL, and AAM: This instruction multiplies an unsigned byte or word by the contents of AL. The unsigned byte or word may be in any one of the general-purpose registers or memory locations. The most significant word of the result is stored in DX, while the least significant word of the result is stored in AX.

The mul instruction, with an eight bit operand, multiplies the al register by the operand and stores the 16 bit result in ax. So

ul operand	(Unsigned)	MUL BL	i.e. AL * BL; Al=25 * BL=04; AX=00 (AH) 64 (AL)
imul operand	(Signed)	IMUL BL	i.e. AL * BL; AL=09 * BL=-2; AL * 2's comp(BL) AL=09 * BL (0EH)=7E; 2's comp (7e)=-82

The aam (ASCII Adjust after Multiplication) instruction, adjust an unpacked decimal value after multiplication. This instruction operates directly on the ax register. It assumes that you've multiplied two eight bit values in the range 0..9 together and the result is sitting in ax (actually, the result will be sitting in al since 9*9 is 81,the largest possible value; ah must contain zero). This instruction divides ax by 10 and leaves the quotient in ah and the remainder in al: mul bl; al=9, bl=9 al*bl=9*9=51H; AX=00(AH) 51(AL); AAM ; first hexadecimal value is converted to decimal value i.e. 51 to 81; al=81D; second convert packed BCD to unpacked BCD, divide AL content by 10 i.e. 81/10 then AL=01, AH =08; AX = 0801

EXAMPLE:

AL 00000101 unpacked BCD 5
BH 00001001 unpacked BCD 9
MUL BH AL x BH; result in AX
AX = 00000000 00101101 = 002DH
AAM AX = 00000100 00000101 = 0405H, which is unpacked BCD for 45.

If ASCII codes for the result are desired, use next instruction OR AX, 3030H Put 3 in upper nibble of

each byte.

AX = 0011 0100 0011 0101 = 3435H, which is ASCII code for 45

The Division Instructions: DIV, IDIV, and AAD

The 80x86 divide instructions perform a 64/32 division (80386 and later only), a 32/16division or a 16/8 division. These instructions take the form:

Div reg For unsigned division

Div mem

Idiv reg For signed division

Idiv mem

The div instruction computes an unsigned division. If the operand is an eight bit operand ,div divides the ax register by the operand leaving the quotient in al and the remainder(modulo) in ah. If the operand is a 16 bit quantity, then the div instruction divides the 32 bit quantity in dx ax by the operand leaving the quotient in ax and the remainder in .

Note: If an overflow occurs (or you attempt a division by zero) then the80x86 executes an INT 0 (interrupt zero).

The aad (ASCII Adjust before Division) instruction is another unpacked decimal operation. It splits apart unpacked binary coded decimal values before an ASCII division operation. The aad instruction is useful for other operations. The algorithm that describes this instruction is

al := ah*10 + al AX=0905H; BL=06; AAD; AX=AH*10+AL=09*10+05=95D;

convert decimal to hexadecimal; 95D=5FH; al=5f;

DIV BL; AL/BL=5F/06; AX=05(AH)0F(AL) ah := 0

EXAMPLE:

AX = 0607H unpacked BCD for 67 decimal CH = 09H, now adjust to binary

AAD Result: AX = 0043 = 43H = 67 decimal

DIV CH Divide AX by unpacked BCD in CH

Quotient: AL = 07 unpacked BCD Remainder:

AH = 04 unpacked BCD Flags undefined after DIV

NOTE: If an attempt is made to divide by 0, the 8086 will do a type 0 interrupt.

CBW-Convert Signed Byte to Signed Word: This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be the sign extension of AL. The CBW operation must be done before a signed byte in AL can be divided by another signed byte with the IDIV instruction. CBW affects no flags.

EXAMPLE:

AX = 00000000 10011011 155 decimal

CBW Convert signed byte in AL to signed word in AX

Result: AX = 11111111 10011011 155 decimal

CWD-Convert Signed Word to Signed Double word: CWD copies the sign bit of a word in AX to all the bits of the DX register. In other words it extends the sign of AX into all of DX. The CWD operation must be done before a signed word in AX can be divided by another signed word with the IDIV instruction. CWD affects no flags.

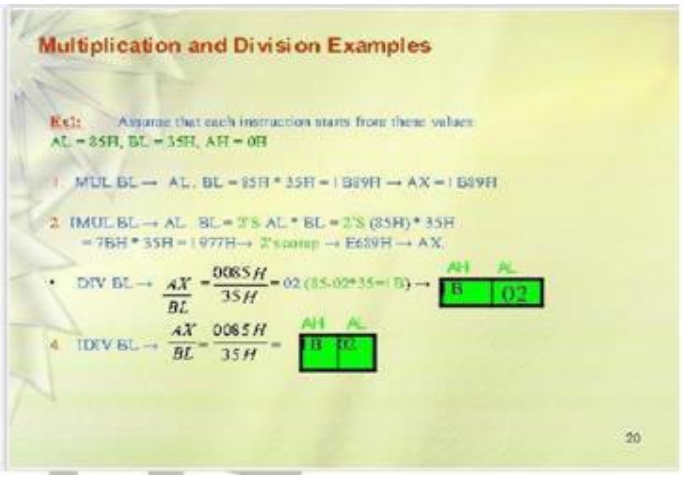
EXAMPLE:

DX = 00000000 00000000

AX = 11110000 11000111 3897 decimal

CWD Convert signed word in AX to signed double word in DX:AX

Result DX = 11111111 11111111
AX = 11110000 11000111 3897 decimal



Logical, Shift, Rotate and Bit Instructions: The 80x86 family provides five logical instructions, four rotate instructions, and three shift instructions. The logical instructions are and, or, xor, test, and not; the rotates are ror, rol, rcr, and rcl; the shift instructions are shl/sal, shr, and sar.

The Logical Instructions: AND, OR, XOR, and NOT: The 80x86 logical instructions operate on a bit-by-bit basis. Except not, these instructions affect the flags as follows:

- They clear the carry flag.
- They clear the overflow flag.
- They set the zero flag if the result is zero, they clear it otherwise.
- They copy the H.O. bit of the result into the sign flag.
- They set the parity flag according to the parity (number of one bits) in the result.
- They scramble the auxiliary carry flag.

The not instruction does not affect any flags.

The AND instruction sets the zero flag if the two operands do not have any ones in corresponding bit positions. AND AX, BX

The OR instruction will only set the zero flag if both operands contain zero. OR AX, BX

The XOR instruction will set the zero flag only if both operands are equal. Notice that the xor operation will produce a zero result if and only if the two operands are equal. Many programmers commonly use this fact to clear a sixteen bit register to zero since an instruction of the form

xor reg16, reg16; XOR AX, AX is shorter than the comparable mov reg,0 instruction.

You can use the and instruction to set selected bits to zero in the destination operand. This is known as masking out data; Likewise, you can use the or instruction to force certain bits to one in the destination operand;

The Shift Instructions: SHL/SAL, SHR, SAR: The 80x86 supports three different shift instructions (shl and sal are the same instruction):shl (shift left), sal (shift arithmetic left), shr (shift right), and sar (shift arithmetic right).

SHL/SAL: These instructions move each bit in the destination operand one bit position to the left the number of times specified by the count operand. Zeros fill vacated positions at the L.O. bit; the H.O.

bit shifts into the carry flag.

The shl/sal instruction sets the condition code bits as follows:

• If the shift count is zero, the shl instruction doesn't affect any flags.
• The carry flag contains the last bit shifted out of the H.O. bit of the operand.
• The overflow flag will contain one if the two H.O. bits were different prior to a single bit shift. The overflow flag is undefined if the shift count is not one.
• The zero flag will be one if the shift produces a zero result.
• The sign flag will contain the H.O. bit of the result.
• The parity flag will contain one if there are an even number of one bits in the L.O. byte of the result.
• The A flag is always undefined after the shl/sal instruction.

The shift left instruction is especially useful for packing data. For example, suppose you have two nibbles in al and ah that you want to combine. You could use the following code to do this:

```
shl ah, 4 ;
or al, ah ;Merge in H.O. four bits.
```

Of course, al must contain a value in the range 0..F for this code to work properly (the shift left operation automatically clears the L.O. four bits of ah before the or instruction).

SHL OPERATION

H.O. four bits of al are not zero before this operation, you can easily clear them with an and instruction:

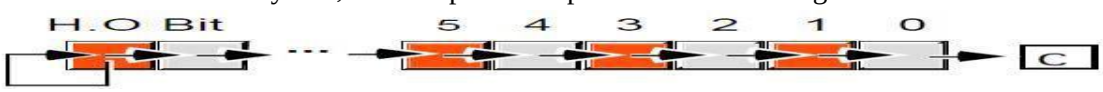
```
shl ah, 4 ;Move L.O. bits to H.O. position.
and al, 0Fh ;Clear H.O. four bits.
or al, ah ;Merge the bits.
```

Since shifting an integer value to the left one position is equivalent to multiplying that value by two, you can also use the shift left instruction for multiplication by powers of two:

```
shl ax, 1 ;Equivalent to AX*2
shl ax, 2 ;Equivalent to AX*4
shl ax, 3 ;Equivalent to AX*8
```

SAR: The sar instruction shifts all the bits in the destination operand to the right one bit, replicating the H.O. bit.

The sar instruction's main purpose is to perform a signed division by some power of two. Each shift to the right divides the value by two. Multiple right shifts divide the previous shifted result by two, so multiple shifts produce the following results:



SAR OPERATION

```
sar ax, 1 ;Signed division by 2 sar
ax, 2 ;Signed division by 4 sar ax,
3 ;Signed division by 8 sar ax, 4
;Signed division by 16 sar ax, 5
;Signed division by 32 sar ax, 6
;Signed division by 64 sar ax, 7
;Signed division by 128 sar ax, 8
;Signed division by 256
```

There is a very important difference between the sar and idiv instructions. The idiv instruction always truncates towards zero while sar truncates results toward the smaller result. For positive results, an arithmetic shift right by one position produces the same result as an integer division by two. However, if the quotient is negative, idiv truncates towards zero while sar truncates towards negative infinity.

SHR: The shr instruction shifts all the bits in the destination operand to the right one bit shifting a zero into the H.O. bit

SHR OPERATION

The shift right instruction is especially useful for unpacking data. shifting an unsigned integer value to the right one position is equivalent to dividing that value by two, you can also use the shift right instruction for division by powers of two:

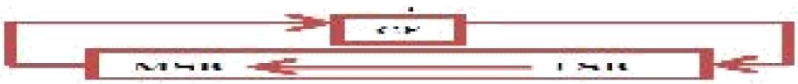
```
shr ax, 1 ;Equivalent to AX/2
shr ax, 2 ;Equivalent to AX/4
shr ax, 3 ;Equivalent to AX/8
shr ax, 4 ;Equivalent to AX/16
```

The Rotate Instructions: RCL, RCR, ROL, and ROR

The rotate instructions shift the bits around, just like the shift instructions, except the bits shifted out of the operand by the rotate instructions re-circulate through the operand. They include rcl(rotate through carry left), rcr(rotate through carry right), rol(rotate left), and ror(rotate right). These instructions all take the forms :rcldest, count roldest, count rcr dest, count ror dest, count

RCL: The rcl(rotate through carry left), as its name implies, rotates bits to the left, through the carry flag, and back into bit zero on the right. The rcl instruction sets the flag bits as follows:

- The carry flag contains the last bit shifted out of the H.O. bit of the operand.
- If the shift count is one, rcl sets the overflow flag if the sign changes as a result of the rotate. If the count is not one, the overflow flag is undefined.
- The rcl instruction does not modify the zero, sign, parity, or auxiliary carry flags.



RCL OPERATION

RCR: The rcr (rotate through carry right) instruction is the complement to the rcl instruction. It shifts its bits right through the carry flag and back into the H.O. bit. This instruction sets the flags in a manner analogous to rcl:

- The carry flag contains the last bit shifted out of the L.O. bit of the operand.
- The rcr instruction does not affect the zero, sign, parity, or auxiliary carry flags.

RCR OPERATION

ROL: The rol instruction is similar to the rcl instruction in that it rotates its operand to the left the specified number of bits. The major difference is that rol shifts its operand's H.O. bit ,rather than the carry, into bit zero. Rol also copies the output of the H.O. bit into the carry flag . The rol instruction sets the flags identically to rcl. Other than the source of the value shifted into bit zero, this instruction behaves exactly like the rcl instruction.

Like shl, the rol instruction is often useful for packing and unpacking data.

ROL OPERATION

ROR: The ror instruction relates to the rcr instruction in much the same way that the rol instruction relates to rcl. That is, it is almost the same operation other than the source of the input bit to the operand. Rather than shifting the previous carry flag into the H.O. bit of the destination operation, ror shifts bit zero into the H.O. bit.

ROR OPERATION

String Instructions: A string is a collection of objects stored in contiguous memory locations. Strings are usually arrays of bytes or words on 8086.**All members of the 80x 86 families support five different string instructions: MOVS, CMPS, SCAS, LODS, AND STOS.**

The string instructions operate on blocks (contiguous linear arrays) of memory. For example, the movs

instruction moves a sequence of bytes from one memory location to another. The cmps instruction compares two blocks of memory. The scas instruction scansa block of memory for a particular value. These string instructions often require three operands, a destination block address, a source block address, and (optionally) an element count. For example, when using the movs instruction to copy a string, you need a source address, a destination address, and a count (the number of string elements to move).The operands for the string instructions include:

- **theSI (source index) register,**
- **the DI (destination index) register,**
- **the** CX (count) register,
- **theAX register, and**
- **the direction flag in the FLAGS register.**

The REP/REPE/REPZ and REPNZ/REPNE Prefixes: The repeat prefixes tell the 80x86 to do a multi-byte string operation. The syntax for the repeat prefix is:

Field:

Label repeat mnemonic operand ; comment

For MOVS:

Rep movs {operands}

For CMPS:

Repe cmps {operands} Repz cmps {operands} Repne cmps {operands} Repnz cmps {operands}

For SCAS:

Repe scas {operands} Repz scas {operands} Repne scas {operands} repnz scas {operands}

For STOS:

epstos {operands}

When specifying the repeat prefix before a string instruction, the string instruction repeats cx times. Without the repeat prefix, the instruction operates only on a single byte,word, or double word.

If the direction flag is clear, the CPU increments si and di after operating upon eachstring element. If the direction flag is set, then the 80x86 decrements si and di after processing eachstring element. The direction flag may be set or cleared using the cld (clear direction flag) and std (setdirection flag) instructions.

The MOVS Instruction: The movsb (move string, bytes) instruction fetches the byte at address ds:si, stores it at address es:di, and then increments or decrements the si and di registers by one. If the rep prefix is present, the CPU checks cx to see if it contains zero. If not, then it moves the byte from ds:si to es:di and decrements the cx register. This process repeats until cx becomes zero. The syntax is :

{REP} MOVSB {REP} MOVSW

The CMPS Instruction: The cmps instruction compares two strings. The CPU compares the string referenced by es:di to the string pointed at by ds:si. Cx contains the length of the two strings (whenusing the rep prefix). The syntax is: {REPE} CMPSB {REPE} CMPSW

To compare two strings to see if they are equal or not equal, you must compare corresponding **elements in a string until they don't match or length of the string cx=0.** Therepe prefix accomplishes this operation. It will compare successive elements in a string as long as they are equal and cx is greater than zero.

The SCAS Instruction: The scas instruction, by itself, compares the value in the accumulator (al or ax) against the value pointed at by es:di and then increments (or decrements) di by one or two. The CPU sets the flags according to the result of the comparison. When using the repne prefix (repeat while not equal),

scas scans the string searching for the first string element which is equal to the value in the accumulator. The scas instruction takes the following forms:{REPNE} SCASB {REPNE} SCASW The STOS

Instruction: The stos instruction stores the value in the accumulator at the location specified byes:di. After storing the value, the CPU increments or decrements di depending upon the state of the direction flag. Its primary use is to initialize arrays and strings to a constant value. {REP} STOSB
{REP} STOSW

The LODS Instruction: The lods instruction copies the byte or word pointed at by ds:si into the al or ax register, after which it increments or decrements the si register by one or two.{REP} LODSB
{REP} LODSW

Flag Manipulation and Processor Control Instructions: These instructions control the functioning of the available hardware inside the processor chip. These are categorized into two types; (a) flag manipulation instructions and (b) machine control instructions.

The flag manipulation instructions directly modify some of the flags of 8086. The machine control instructions control the bus usage and execution. The flag manipulation instructions and their functions are as follows:

CLC - Clear carry flag CMC - Complement carry flag STC - Set carry flag
CLD - Clear direction flag STD - Set direction flag CLI - Clear interrupt flag
STI - Set interrupt flag

These instructions modify the carry(CF), direction(DF) and interrupt(IF) flags directly. The DF and IF, which may be modified using the flag manipulation instructions, further control the processor operation; like interrupt responses and auto-increment or auto-decrement modes.

The machine control instructions supported by 8086 and 8088 are listed as follows along with their functions. These machine control instructions do not require any operand.

WAIT - Wait for Test input pin to go low HLT - Halt the processor NOP No Operation ESC - Escape to external device like NDP (numeric co-processor) LOCK Bu lock instruction prefix.

After executing the HLT instruction, the processor enters the halt state. The two ways to pull it out of the halt state are to reset the processor or to interrupt it.

When NOP instruction is executed, the processor does not perform any operation till 4 clock cycles, except incrementing the IP by one. It then continues with further execution after 4 clock cycles.

ESC instruction when executed, frees the bus for an external master like a coprocessor or peripheral devices.

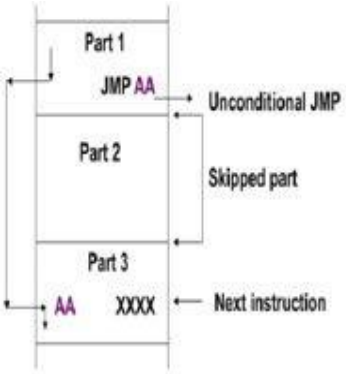
The LOCK prefix may appear with another instruction. When it is executed, the bus access is not allowed for another master till the lock prefixed instruction is executed completely. This instruction is used in case of programming for multiprocessor systems.

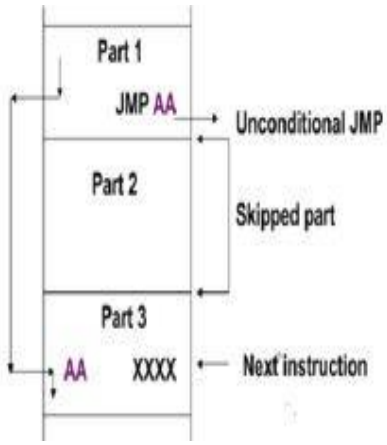
The WAIT instruction when executed holds the operation of processor with the current status till the logic level on the TEST pin goes low. The processor goes on inserting WAIT states in the instruction cycle, till the TEST pin goes low. Once the TEST pin goes low, it continues further execution.

Program Flow Control Instructions: The control transfer instructions are used to transfer the control from one memory location to another memory location. In 8086 program control instructions belong to three groups: unconditional transfers, conditional transfers, and subroutine call and return instructions.

Unconditional Jumps: The jmp (jump) instruction unconditionally transfers control to another point in the program. Intra segment jumps are always between statements in the same code segment. Intersegment jumps can transfer control to a statement in a different code segment.

JMP Address





Unconditional jump Conditional jump

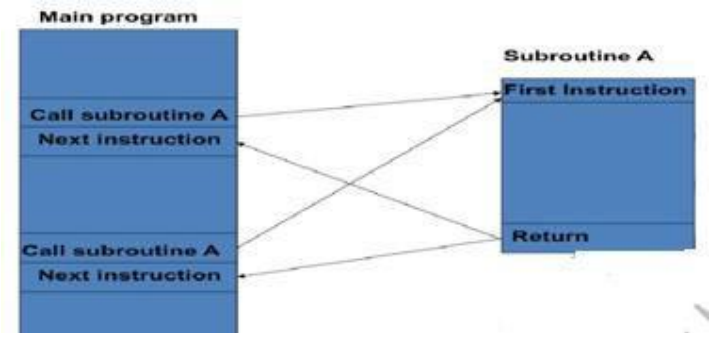
Conditional Jump: The conditional jump instructions are the basic tool for creating loops and other conditionally executable statements like the if…..then statement. The conditional jumps test one or more bits in the status register to see if they match some particular pattern. If the pattern matches, control transfers to the target location. If the condition fails, the CPU ignores the conditional jump and execution continues with the next instruction. Some instructions, for example, test the conditions of the sign, carry, overflow and zero flags.

Definition	Description	Condition
Jump Based on Unsigned Data		
JE / JZ	Jump equal or jump zero	Z=1
JNE / JNZ	Jump not equal or jump not zero	Z=0
JA / JNBE	Jump above or jump not below/ equal	C=0 & Z=0
JAE / JNB	Jump above/ equal or jump not below	C=0
JB / JNAE	Jump below or jump not above/ equal	C=1
JBE / JNA	Jump below/ equal or jump not above	C=1 or Z=1
Jump Based on Signed Data		
JE / JZ	Jump equal or jump zero	Z=1
JNE / JNZ	Jump not equal or jump not zero	Z=0
JG / JNLE	Jump greater or jump not less/ equal	N=0 & Z=0
JGE / JNL	Jump greater/ equal or jump not less	N=0
JL / JNGE	Jump less or jump not greater/ equal	N=1
JLE / JNG	Jump less/ equal or jump not greater	N=1 or Z=1
Arithmetic Jump		
JS	Jump sign set	N=1
JNS	Jump no sign set	N=0
JC	Jump carry set	C=1
JNC	Jump no carry set	C=0
JO	Jump overflow set	O=1
JNO	Jump not overflow set	O=0
JP / JPE	Jump parity even	P=1
JNP / JPO	Jump parity odd	P=0

Loop Instruction:

- These instructions are used to repeat a set of instructions several times.
- Format: LOOP Short-Label
- Operation: (CX) ← (CX)-1
- Jump is initialized to location defined by short label if CX≠0. Otherwise, execute next sequential instruction.
- Instruction LOOP works with respect to contents of CX. CX must be preloaded with a count that represents the number of times the loop is to be repeat.
- Whenever the loop is executed, contents at CX are first decremented then checked to determine if they are equal to zero.
- If CX=0, loop is complete and the instruction following loop is executed.
- If CX ≠ 0, content return to the instruction at the label specified in the loop instruction.
- LOOP AGAIN is almost same as: DEC CX, JNZ AGAIN

SUBROUTINE & SUBROUTINE HANDILING INSTRUCTIONS: CALL, RET



- A subroutine is a special segment of program that can be called for execution from any point in a program.
- An assembly language subroutine is also referred to as a "procedure".
- Whenever we need the subroutine, a single instruction is inserted in to the main body of the program to call subroutine.
- Transfers the flow of the program to the procedure.
- CALL instruction differs from the jump instruction because a CALL saves a return address on the
  - stack.
- The return address returns control to the instruction that immediately follows the CALL in a program when a RET instruction executes.
- To branch a subroutine the value in the IP or CS and IP must be modified.
- After execution, we want to return the control to the instruction that immediately follows the one called the subroutine i.e., the original value of IP or CS and IP must be preserved.
- Execution of the instruction causes the contents of IP to be saved on the stack. (this time (SP) ← (SP) -2)
- A new 16-bit (near-proc, mem16, reg16 i.e., Intra Segment) value which is specified by the instructions operand is loaded into IP.
- Examples: CALL 1234H
  CALL BX
  CALL [BX]

Return Instruction: RET instruction removes an address from the stack so the program returns to the instruction following the CALL

- Every subroutine must end by executing an instruction that returns control to the main program. This is the return (RET) instruction.
- By execution the value of IP or IP and CS that were saved in the stack to be returned back to their corresponding registers. (this time (SP) ← (SP)+2)

MACROS: The macro directive allows the programmer to write a named block of source statements, then use that name in the source file to represent the group of statements. During the assembly phase, the assembler automatically replaces each occurrence of the macro name with the statements in the macro definition.

Macros are expanded on every occurrence of the macro name, so they can increase the length of the executable file if used repeatable. Procedures or subroutines take up less space, but the increased overhead of saving and restoring addresses and parameters can make them slower. In summary, the advantages and disadvantages of macros are,

Advantages

- Repeated small groups of instructions replaced by one macro
- Errors in macros are fixed only once, in the definition
- Duplication of effort is reduced
- In effect, new higher level instructions can be created
- Programming is made easier, less error prone
- Generally quicker in execution than subroutines

Disadvantages

In large programs, produce greater code size than procedures

When to use Macros

- To replace small groups of instructions not worthy of subroutines
- To create a higher instruction set for specific applications
- To create compatibility with other computers
- To replace code portions which are repeated often throughout the program

UNIT-III
(ARCHITECTURE OF 8086 & INTERFACING)

Syllabus: Pin diagram of 8086-minimum mode and maximum mode of operation, Timing diagram, memory interfacing to 8086 (static RAM and EPROM). Need for DMA, DMA data transfer method, interfacing with 8237/8257.

INTRODUCTION

This unit explains how to design and implement an 8086 based microcomputer system. To design an 8086 based system, it is necessary to know how to interface the 8086 microprocessor with memory and input and output devices. Due to the mismatch in the speed between the microprocessor and other devices, a set of latches and buffers are required to interface the microprocessor with other devices. In this unit, you will learn about the way in which address/data buses, latches and buffers are used in the process of interfacing. To understand the interfacing principles and concepts it is necessary to learn the various types of bus cycles and bus timings. Overall, this unit makes you to understand how 8086 microprocessor is interfaced with memory and peripherals and how an 8086 based microcomputer system works.

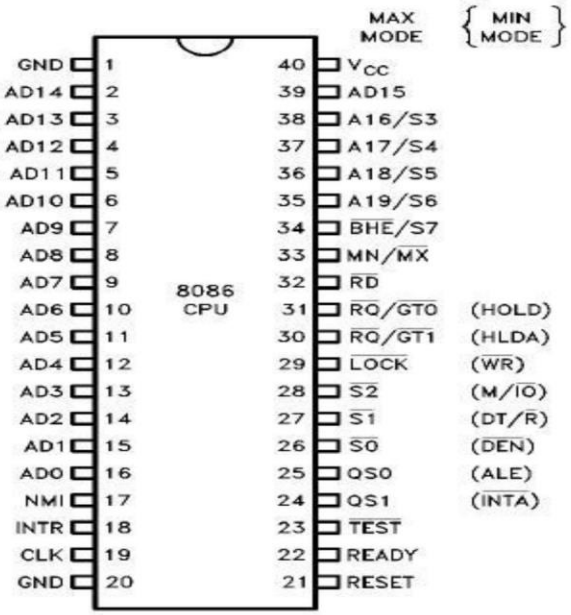
PIN DIAGRAM OF 8086 MICROPROCESSOR

The Microprocessor 8086 is a 16-bit MICROPROCESSOR available in different clock rates (5, 8, 10 MHz) and packaged in a 40 pin DIP or plastic package. The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

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

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

The figure below shows the pins/signals of 8086 processor. Here the pins within the brackets (minimum mode pins) are minimum mode pins.



Signal description:

The 8086 signals can be categorized in three groups.

- The first are the signal having common functions in minimum as well as maximum mode.
- The second are the signals which have special functions for minimum mode.
- Third are the signals which have special functions for maximum mode.

✓ **The following signal descriptions are common for both modes:**

Vcc: It requires +5V single power supply for the operation of the internal circuit.

GND: ground for the internal circuit.

AD15-AD0: These are the time multiplexed memory I/O address and data lines. These lines serve two functions. The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D15 is the MSB and D0 LSB. When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

A19/S6, A18/S5, A17/S4, and A16/S3: These are the time multiplexed address and status lines. During T1 these are the most significant address lines for memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2, T3, Tw and T4.

The status of the interrupt enable flag bit is updated at the beginning of each clock cycle. The status is displayed on $\mathbf{S_5}$ pin.

The **S4** and **S3** combinedly indicate which segment register is presently being used for memory accesses as in below fig.

The last status bit **S6** is always at the logic 0 level.

The address bits are separated from the status bit using latches controlled by the ALE signal.

S_4	S_3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

$\overline{BHE}$**/S7:** The bus high enable 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 transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. $\overline{BHE}$ is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus.

BHE	A_0	Indication
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address
1	1	None

$\overline{RD}$ (Read): This signal on low indicates the peripheral that the processor is performing s memory or I/O read operation. $\overline{RD}$ 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 device or memory that they have completed the data transfer. This signal is provided by an external clock generator device and can be supplied by the memory or I/O subsystem to signal the 8086 when they are ready to permit the data transfer to be completed.

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.

INTR: INTR is an input to the 8086 that can be used by an external device to signal that it needs to be serviced. Logic 1 at INTR represents an active interrupt request. When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the $\overline{INTA}$ output.

$\overline{TEST}$: This input is examined by a 'WAIT' instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state.

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.

RESET: This input causes the processor to terminate the current activity and start execution from FFFF0H.

MN/$\overline{MX}$: The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.

- ✓ **The following pin functions are for the minimum mode operation of 8086:**

M/$\overline{IO}$: This is a status line logically equivalent to S2 in maximum mode. The logic level of M/$\overline{IO}$ tells external circuitry whether a memory or I/O transfer is taking place over the bus. When it is low, it indicates the processor is having an I/O operation, and when it is high, it indicates that the processor is having a memory operation.

$\overline{WR}$: The signal write $\overline{WR}$ indicates that a write bus cycle is in progress. The 8086 switches $\overline{WR}$ to logic 0 to signal external device that valid write or output data are on the bus.

$\overline{INTA}$: (Interrupt Acknowledge)-This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.

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/$\overline{R}$ – Data Transmit/Receive: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

$\overline{DEN}$ **– 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.

HOLD and HLDA: The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals. 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 cycle.

The following pin functions are applicable for maximum mode operation of 8086:
$\overline{S2}$,$\overline{S1}$, **and** $\overline{S0}$ **– Status Lines**: These are the status lines which reflect the type of operation, being carried out by the processor.

Status Inputs			CPU Cycles
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

LOCK: This output pin indicates that other system bus master will be prevented from gaining the system bus, while the LOCK signal is low.

QS1, QS0 – Queue Status: These lines give information about the status of the code-prefetch queue. Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS0 and QS1. Together they form a 2-bit queue status code, QS1QS0. Following table shows the four different queue status.

QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

RQ0/0GT, RQ1/GT1 – Request/Grant: These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle. Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1.

OPERATING MODES OF 8086

There are two modes of operation for Intel 8086 namely the minimum mode and the maximum mode. When only one 8086 microprocessor is to be used in a micro computer system the 8086 is used in the minimum mode of operation. In this mode the microprocessor issues the control signals required by memory and I/O devices. In a multi processor system it operates in the maximum mode. In case of maximum mode of operation control signals are issued by Intel 8288 bus controller which is used with 8086 for this purpose. The level of the pin $MN/\overline{MX}$ decides the operating mode of 8086. When $MN/\overline{MX}$ is high the microprocessor operates in a minimum mode. When it is low the microprocessor operates in the maximum mode. From pin 24 to 31 issue two different sets of signals. One set of signals is issued when the microprocessor is operating in the minimum mode. The other sort of signal is issued when the microprocessor is operating in the maximum mode. Thus the pins from 24-31 have alternate functions.

TIMINGS:

Timing plays a crucial role, not only in sports like cricket_but also in digital electronic equipments like microprocessors. Timing and Timing diagram plays a vital role in microprocessors. The timing diagram is the diagram which provides information about the various conditions of signals such as high/low, when a machine cycle is being executed. Without the knowledge of timing diagram it is not possible to match the peripheral devices to the microprocessors. These peripheral devices includes memories, ports etc. Such devices can only be matched with microprocessors with the help of timing diagram.

Before dealing with timing diagram, we have to make ourselves familiar with certain terms.

Machine cycle: A basic microprocessor operation such as reading a byte from memory or writing a byte to a port is called a machine cycle (Bus cycle). A machine cycle consists of at least 4 clock cycles/ clock states (T-states) for accessing the data.

Instruction cycle: The time a microprocessor requires to fetch and execute an entire instruction is referred to as an instruction cycle. An instruction cycle consists of one or more machine cycles.

T-state: T-state is nothing but one subdivision of the operation performed in one clock period. These subdivisions are internal state of the microprocessor synchronized with system clock.

So, an instruction cycle is made up of machine cycles, and a machine cycle is made up of states. The time for the state is determined by the frequency of the clock signal.

GENERAL BUS OPERATION CYCLES:

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 transreceivers, when ever 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 above figs shows the signal activities on the 8086 microcomputer buses during simple read and write operations. The first line look at is the clock waveform, CLK, at the top. This represent s the crystal controlled clock signal sent to the 8086 from an external clock generator device such as the 8284. One clock cycle of this clock is called a state. The time interval labeled T1 in the figure is an example of a state. Different versions of the 8086 have maximum clock frequencies of between 5 MHz and 10 MHz, so the minimum time for one sate will be between 100 and 200ns, depending on the part use and the crystal used.

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 $\overline{S0}$, $\overline{S1}$ and $\overline{S2}$ are used to indicate the type of operation. Status bits S3 to S7 are multiplexed with higher order address bits and the $\overline{BHE}$ signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.

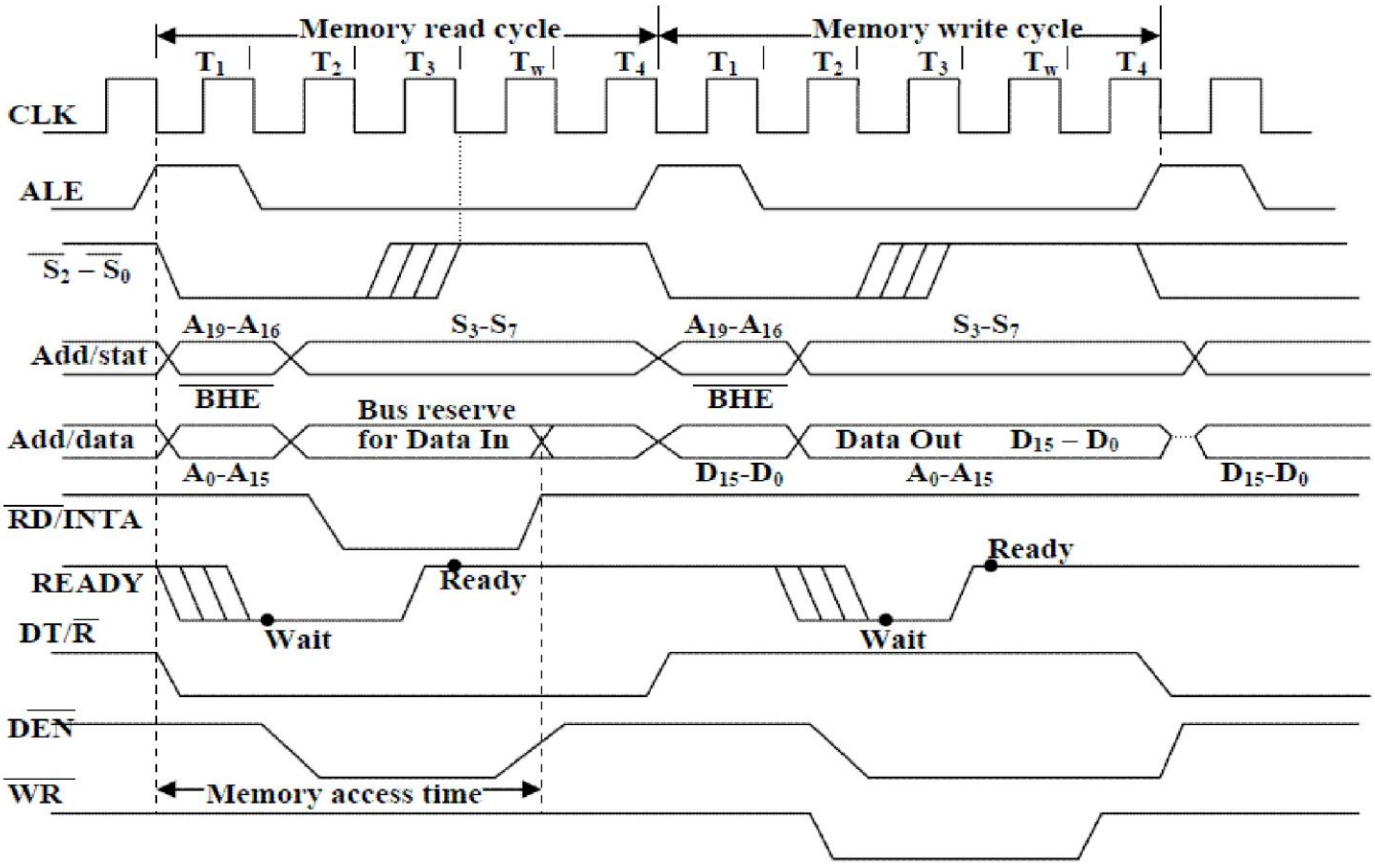


Fig. General bus operation timing diagram

8086 Bus activities during a Read machine cycle:

Fig above (left half portion) shows the timing diagram of 8086 read machine cycle with WAIT state. The clock (CLK) signal is obtained from the clock-generator 8284. Each cycle of the clock is referred to as a state. Minimum number of states to access a data is four. They are T1, T2, T3, and T4 states.

During T1 state of a read machine cycle an 8086 first asserts the M/$\overline{IO}$ signal. It will assert this signal high if it is going to read from memory during memory read cycle and it will assert M/$\overline{IO}$ low if it is going to do a read from an Input port during its read cycle. The timing diagram in fig shows two lines for the M/$\overline{IO}$ signal, because the signal may be going LOW or going HIGH for a read cycle. The point where the two lines cross indicate the time at which the signal becomes valid for this machine cycle.

After asserting M/$\overline{IO}$, the 8086 sends out a high on the address latch enable signal, ALE. The microprocessor sends out on AD0-AD15, A16 through A19 and $\overline{BHE}$ lines, and the address of the memory location that it wants to read. Since the latches are enabled by ALE being high, this address information passes through the latches to their outputs. The 8086 then makes the ALE output low. This disables the latches (8282) and holds the address information latched on the latch outputs. The address information latched on the latch outputs can now be used to select the desired memory or port location.

In the timing diagram, the first point at which the two ($AD_0 - AD_{15}$) cross represents the time at which the 8086 has put a valid address on these lines. Two lines DO NOT indicate that all 16 lines are going high or going low at this point. The crossed lines indicate the time at which a valid address is on the bus.

Since the address information is now held on the latch, the 8086 does not need to send it out any more. As shown in fig. 12 the 8086 floats the AD0 - AD15 lines so that they can be used to input data from memory or from a port. At about the same time the 8086 also remove the $\overline{BHE}$ and A16-A19 information from the upper lines and sends out some status information on these lines.

The 8086 is now ready to read data from the addressed memory locations or port. During T2-state the 8086 asserts its $\overline{RD}$ signal low. This signal is used to enable the addressed memory device or port device.

At the end of T3 state the microprocessor makes the $\overline{RD}$ signal high and reads the data available on the data bus, provided the READY input signal is high. It is the duty of the external circuit to see that valid data is made available on the data bus.

If the READY input pin is not high at the sampled time in a machine cycle, the 8086 will insert one or more WAIT states between T3 and T4 states in that machine cycle. An external hardware device is set up to pulse READY low before the rising edge of the clock in T2 state. After the 8086 finishes T3 of the machine cycle, it enters a WAIT state.

If the READY input is still low at the end of a WAIT state, then the 8086 will insert another WAIT state. The 8086 will continue inserting WAIT states until the READY input is sampled high again. If the READY input is

sampled high again during T3 or during the WAIT state, the microprocessor comes out of the WAIT state and will initiate T4 of the machine cycle.

The $\overline{DEN}$ signal is used to enable bi-directional buffers on the data bus. The data enable signal, $\overline{DEN}$, from the 8086 will enable the data buffer when it is asserted LOW. The data transmit / receive signal DT/$\overline{R}$ from the 8086 is used to specify the direction in which the buffers are enabled. When DT/$\overline{R}$ is asserted high, the buffers will, if enabled by $\overline{DEN}$, transmit data from the 8086 to Memory or I/O ports. When DT/$\overline{R}$ is asserted low, the buffers, if enabled by $\overline{DEN}$, will allow data to be received from Memory or I/O ports of the 8086. DT/$\overline{R}$ is asserted during T1 of the machine cycle. The $\overline{DEN}$ is asserted after the 8086 finishes using the data bus to send the lower 16 address bits

8086 Bus activities during write machine cycle:

The 8086 write operation is very similar to the read cycle. During T1 of a write machine cycle the 8086 asserts M/$\overline{IO}$ low if the write is going to a port and it asserts M/$\overline{IO}$ high if the write is going to memory. At about the same time the 8086 raises ALE high to enable the address latches. The 8086 then assert $\overline{BHE}$ and on the lines AD0 - AD19, it output the address that it will be writing to. When writing to a port, line A16 - A19 will always be low, because the 8086 only sends out 16-bits port addresses. The 8086 brings ALE low again to latch the address on the outputs of the latches. In addition to holding the address, the latches also function as buffers for the address lines. After the address information is latched, the 8086 remove the address information from AD0 - AD15 and outputs the desired data on these lines.

If the READY input is sampled LOW by the 8086 before or during T2 of the machine cycle, the 8086 will insert a WAIT state after T3. If the READY input is sampled high before the end of the WAIT state, the 8086 will go on with state T4 as soon as it completes the WAIT state. The 8086 will continue to insect wait states for as long as the READY is sampled low just before the end of each WAIT state.

MINIMUM MODE 8086 SYSTEM AND TIMINGS

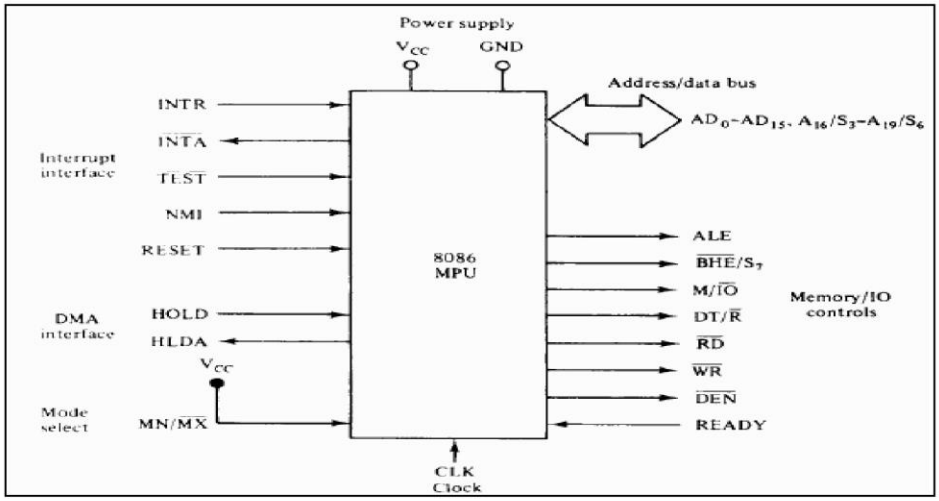


Fig.1: Pin diagram of minimum mode 8086 system

K SUDHAKAR Unit-3

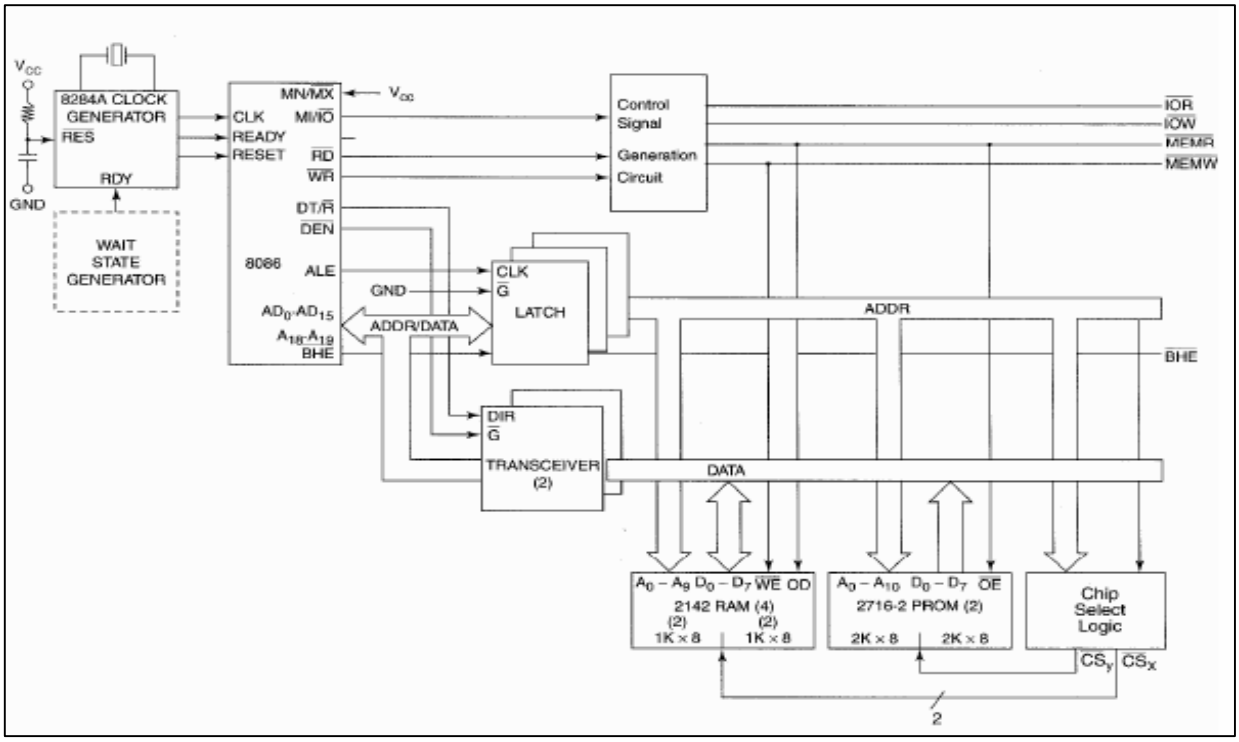


Fig.2: Minimum mode 8086 system configuration

•In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/$\overline{MX}$ pin to logic 1. 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, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

•Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

•Transreceivers are the bidirectional buffers and sometimes they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals. They are controlled by two signals namely, $\overline{DEN}$ and DT/$\bar{R}$. The DT/$\bar{R}$ signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.

•Usually, EPROM is used for monitor storage, while RAM for user's program storage. A system may contain I/O devices.

•The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.

•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.**

Read Cycle:

The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / $\overline{IO}$ signal. During the negative going edge of this signal, the valid address is latched on the local bus.

•The $\overline{BHE}$ and A0 signals address low, high or both bytes. From T1 to T4 , the M/$\overline{IO}$ signal indicates a memory or I/O operation.

•At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read ($\overline{RD}$) control signal is also activated in T2.
•The read ($\overline{RD}$) signal causes the address device to enable its data bus drivers. After $\overline{RD}$ 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.

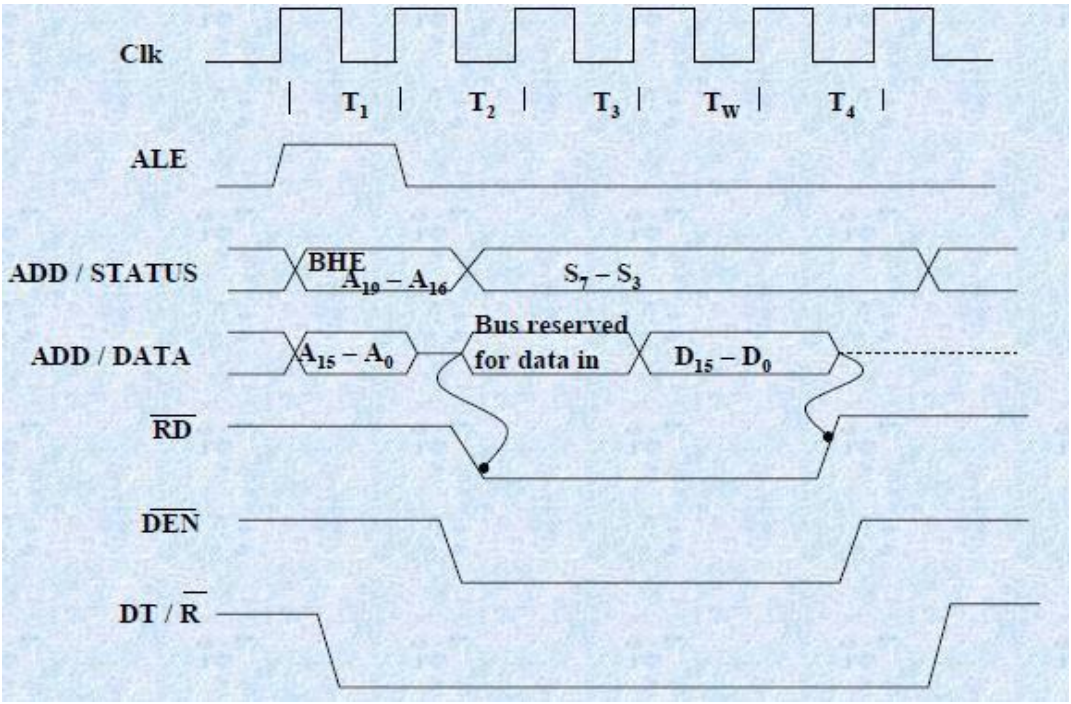


Fig. Read cycle timing diagram for minimum mode operation

Write Cycle:

•A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{IO}$ signal is again asserted to indicate a memory or I/O operation. In T2, after sending the address in T1, the processor sends the data to be written to the addressed location.
•The data remains on the bus until middle of T4 state. The $\overline{WR}$ becomes active at the beginning of T2 (unlike $\overline{RD}$ is somewhat delayed in T2 to provide time for floating).
•The $\overline{BHE}$ and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.

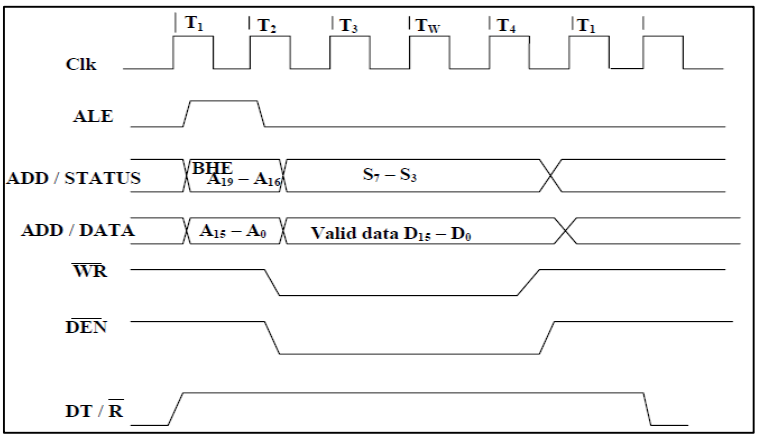


Fig. Write cycle timing diagram for minimum mode operation

The M/$\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ signals indicate the type of data transfer as specified in table below.

M / $\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Transfer Type
0	0	1	I / O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

MAXIMUM MODE 8086 SYSTEM AND TIMINGS

When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment. By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program. Usually in this type of system environment, there are some system resources that are common to all processors. They are called as global resources. There are also other resources that are assigned to specific processors. These are known as local or private resources. Coprocessor also means that there is a second processor in the system. In this two processor does not access the bus at the same time. One passes the control of the system bus to the other and then may suspend its operation. In the maximum-mode 8086 system, facilities are provided for implementing allocation of global resources and passing bus control to other microprocessor or coprocessor.

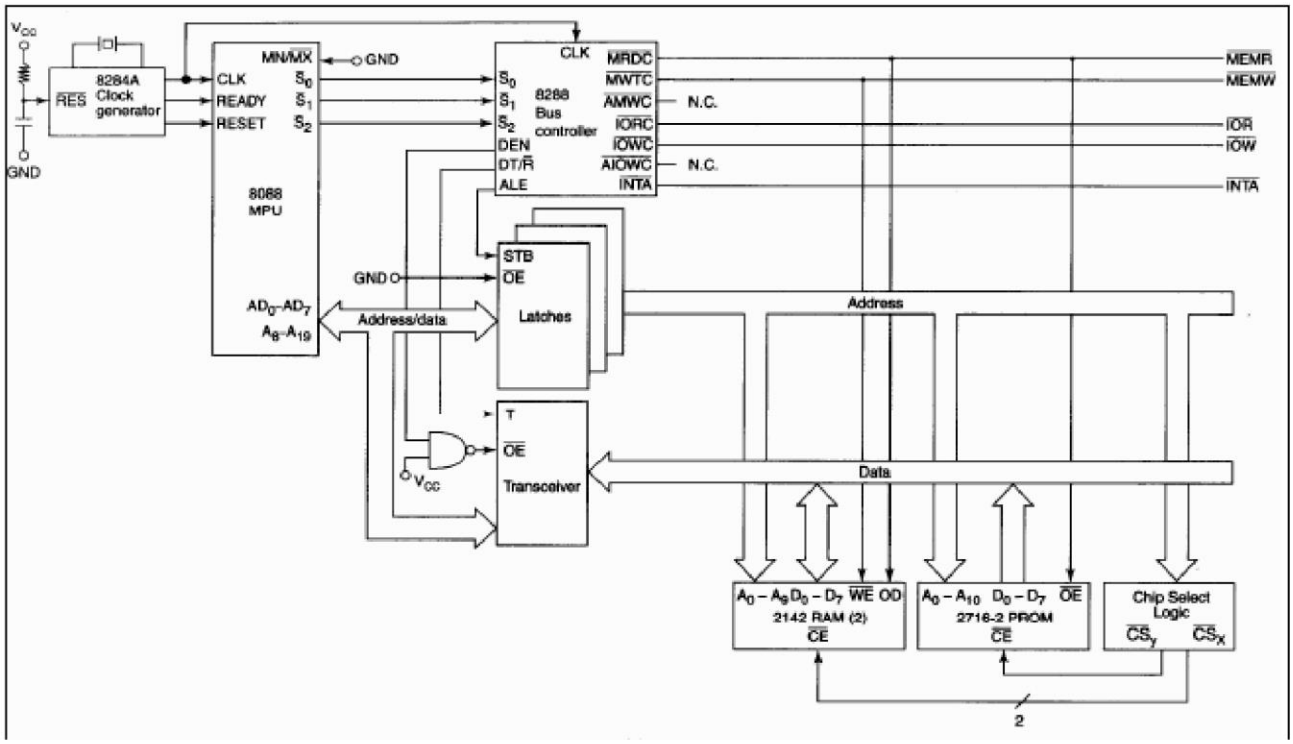


Fig: Maximum mode 8086 system configuration

K SUDHAKAR Unit-3

•In the maximum mode, the 8086 is operated by strapping the MN/$\overline{MX}$ pin to ground. In this mode, the processor derives the status signal $\overline{S2}$, $\overline{S1}$, $\overline{S0}$. Another chip called bus controller derives the control signal using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration. The components in the system are same as in the minimum mode system.

•The basic function of the bus controller chip IC8288 is to derive control signals like $\overline{RD}$ and $\overline{WR}$ (for memory and I/O devices), $\overline{DEN}$, DT/$\overline{R}$, ALE etc. using the information by the processor on the status lines.

•The bus controller chip has input lines $\overline{S2}$, $\overline{S1}$, $\overline{S0}$ and CLK. These inputs to 8288 are driven by MICROPROCESSOR.

•It derives the outputs ALE, $\overline{DEN}$, DT/$\overline{R}$, $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$ and $\overline{AIOWC}$. The AEN, IOB and CEN pins are specially useful for multiprocessor systems.

•AEN and IOB are generally grounded. CEN pin is usually tied to +5V. 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 cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.

•$\overline{INTA}$ pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

•$\overline{IORC}$, $\overline{IOWC}$ 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 address port.

•The $\overline{MRDC}$, $\overline{MWTC}$ are memory read command and memory write command signals respectively and may be used as memory read or write signals.

•All these command signals instructs the memory to accept or send data from or to the bus.

•For both of these write command signals, the advanced signals namely $\overline{AIOWC}$ and $\overline{AMWTC}$ are available.

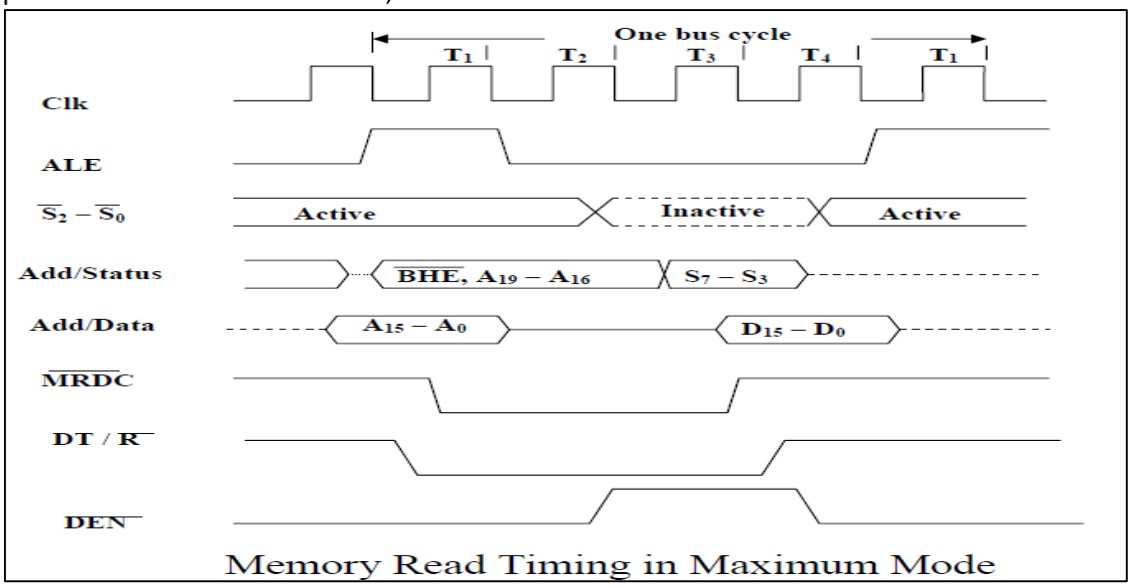
•Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.

•R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.

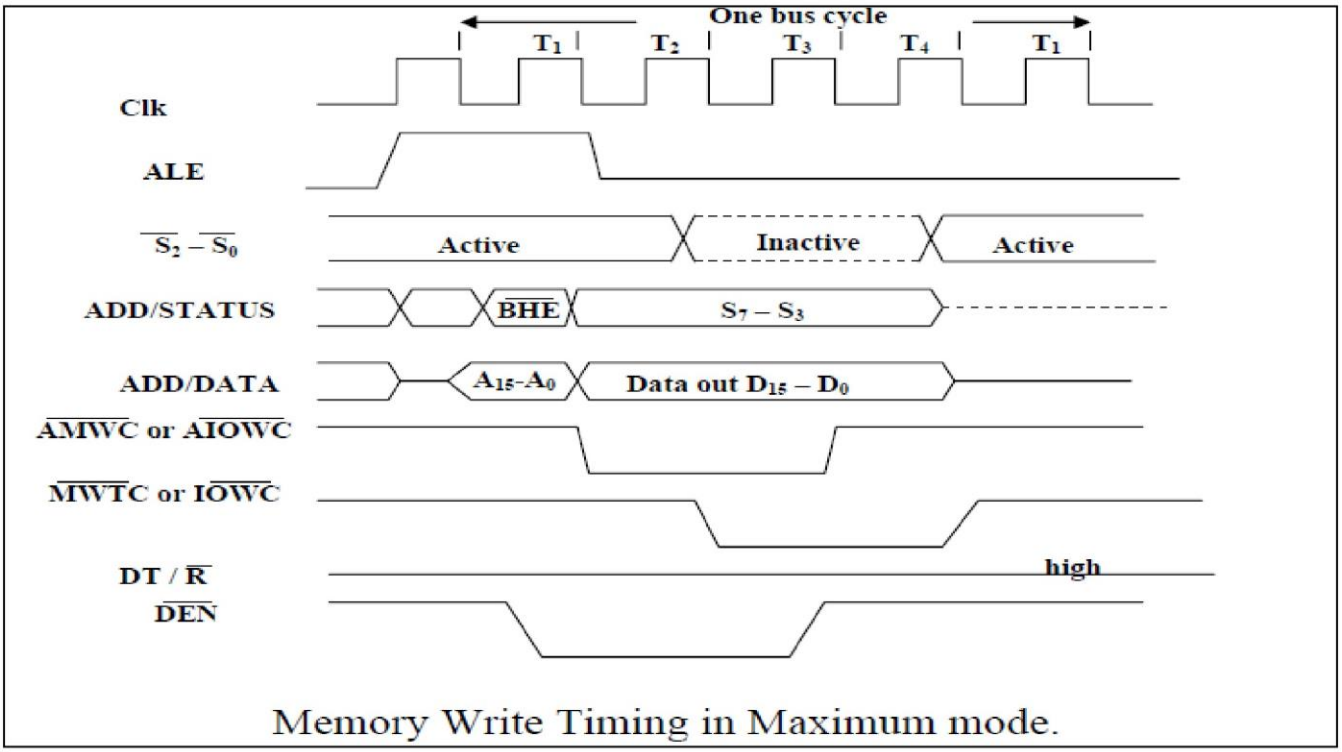
•In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.

•The status bit S0 to S2 remains active until T3 and become passive during T3 and T4.

•If reader input is not activated before T3, wait state will be inserted between T3 and T4.



Memory Read Timing in Maximum Mode



Memory Write Timing in Maximum mode.

ADDRESS AND DATA BUS

The BIU has a combined address and data bus, commonly referred to as a time-multiplexed bus. Time multiplexing address and data information makes the most efficient use of device package pins. A system with address latching provided within the memory and I/O devices can directly connect to the address/data bus. The local bus can be demultiplexed with a single set of address latches to provide non-multiplexed address and data information to the system.

The programmer views the memory or I/O address space as a sequence of bytes. Memory space consists of 1 Mbyte, while I/O space consists of 64 Kbytes. Any byte can contain an 8-bit data element, and any two consecutive bytes can contain a 16-bit data element (identified as a word). The discussions in this section apply to both memory and I/O bus cycles. For brevity, memory bus cycles are used for examples and illustration.

The memory address space on a 16-bit data bus is physically implemented by dividing the address space into two banks of up to 512 Kbytes each (see Figure 1). One bank connects to the lower half of the data bus and contains even-addressed bytes (A0=0). The other bank connects to the upper half of the data bus and contains odd-addressed bytes (A0=1). Address lines A19:1 selects a specific byte within each bank. A0 and Byte High Enable ($\overline{BHE}$) determine whether one bank or both banks participate in the data transfer.

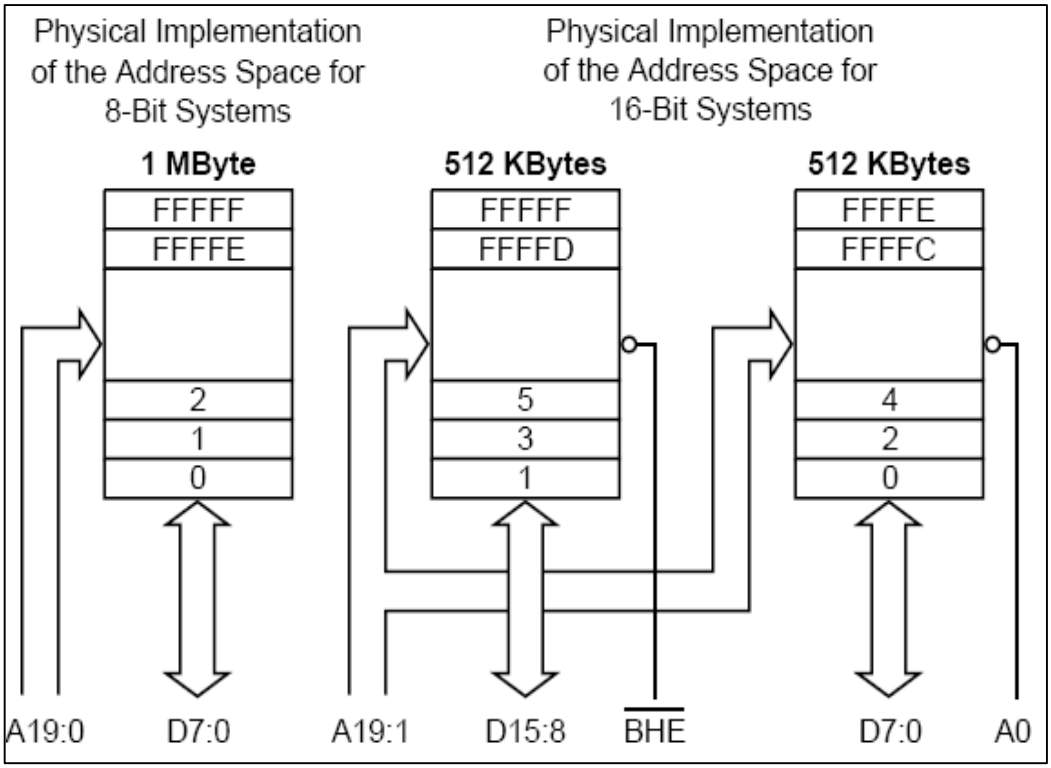


Figure 1. Physical Databus Models

Byte transfers to even addresses transfer information over the lower half of the data bus (see Figure-2a). A0 low enables the lower bank, while BHE high disables the upper bank. The data value from the upper bank is ignored during a bus read cycle. BHE high prevents a write operation from destroying data in the upper bank. Byte transfers to odd addresses transfer information over the upper half of the data bus (see Figure -2b).BHE low enables the upper bank, while A0 high disables the lower bank. The data value from the lower bank is ignored during a bus read cycle. A0 high prevents a write operation from destroying data in the lower bank. To access even-addressed 16-bit words (two consecutive bytes with the least-significant byte at an even address), information is transferred over both halves of the data bus (see Figure-3).

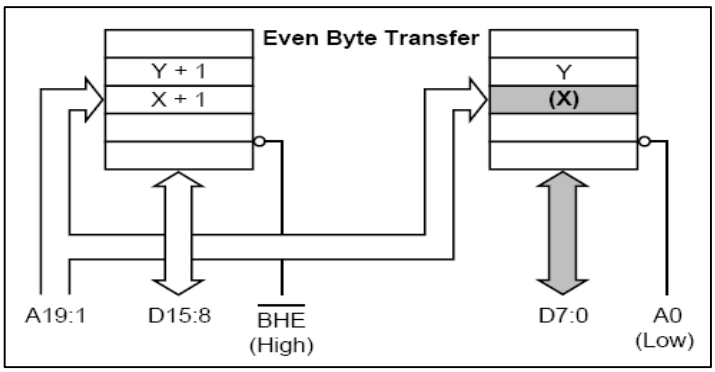


Figure.2a: 16-bit Data Bus Byte Transfers

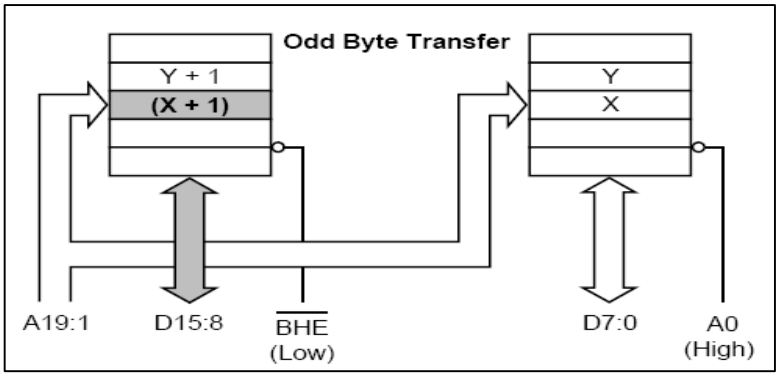


Figure.2b: 16-bit Data Bus Byte transfers

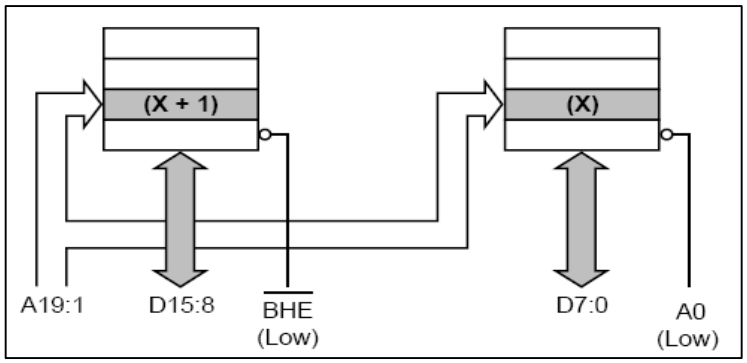


Figure-3: 16-Bit Data Bus Even Word Transfers

A19:1 select the appropriate byte within each bank. A0 and $\overline{BHE}$ drive low to enable both banks simultaneously. Odd-addressed word accesses require the BIU to split the transfer into two byte operations (see Figure-4). The first operation transfers data over the upper half of the bus, while the second operation transfers data over the lower half of the bus. The BIU automatically executes the two-byte sequence whenever an odd-addressed word access is performed.

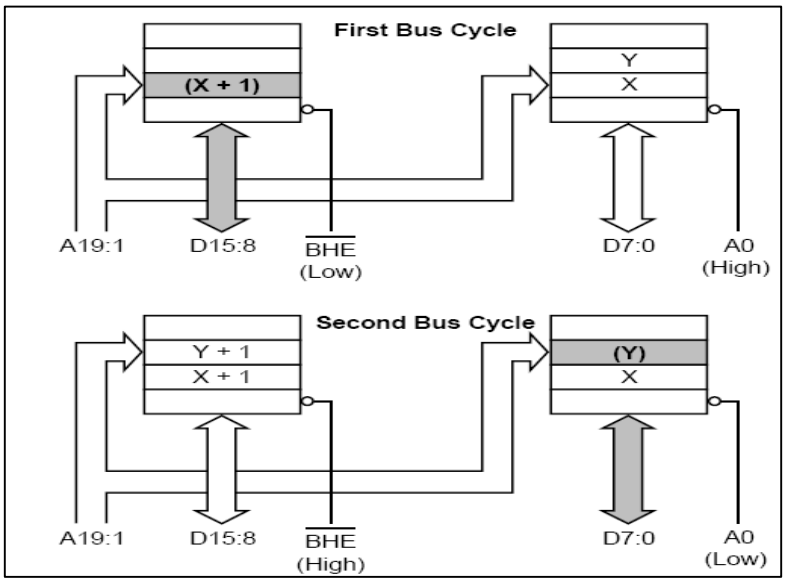


Figure -4 16-Bit data Bus Odd Word Transfers

MEMORY INTERFACING:

8086 Memory Banks

8086 has a 20 bit address bus and hence it can address 2^{20} or 1,048,576 addresses. In each location a byte is stored. So when a word is stored in the memory, it is stored in two consecutive memory locations. Strictly speaking, both memory read and memory write operations require more than one memory cycle. If we want 8086 to complete memory read and memory write operations to be completed with one machine cycle, the memory is to be organized in the form of two banks. Each bank will have 524,288 bytes each.

One memory bank contains all the even addressed locations like 00000, 00002 and 00004. The data lines of this bank are connected to the lower eight data lines, D0 through D7 of the 8086. The other memory bank has all the odd addressed locations like 00001, 00003 and 00005. The data lines of this bank are connected to the upper eight data lines, D8 through D15 of the 8086. Address line A0 is used as part of the enabling for memory in the lower bank. Address lines A1 through A19 are used to select the desired memory device in the bank to address the desired byte in the service. These address lines from A1 through A19 are also used to access a particular location in the upper bank. An additional signal called Bus High Enable (BHE – Active Low) is used to enable the upper memory bank. An external latch, strobed by ALE, grabs the BHE (Active Low) signal that holds it stable for the rest of the machine cycle. The following table shows the required logic levels on the BHE (Active Low) and A0 signals for various types of memory accesses.

Address	Data type	Bhe (active low)	A0	Bus cycles	Data lines used
0000	BYTE	1	0	ONE	D0-D7
0000	WORD	0	0	ONE	D0-D15
0001	BYTE	0	1	ONE	D7-D15
0001	WORD	0 1	1 0	FIRST SECOND	D0-D7 D7-D15

Case 1

Read/Write a byte form/to an even address – A0 will be low and BHE (Active Low) will be high – Byte is transferred to/from low bank through D0-D7

Example – MOV AH, DS: BYTE PTR [0000]

Case 2

Similar to case 1 except the word access instead of the byte access – Both A0 and BHE (Active Low) will be asserted low – Low byte of the word through D0-D7 and high byte of the word through D8-D15

K SUDHAKAR Unit-3

Example – MOV AX,DS:WORD PTR[0000]

Case 3

Read/Wrire a byte from/to an odd address – A0 will be high and BHE (Active Low) will be asserted low – Low bank is disabled and high bank is enabled – Byte is transferred through D0-D7

Example – MOV AL,DS:BYTE PTR[0001]

Case 4

Read/Write a word from/to an odd address – 8086 requires two bus cycles – During the first machine cycle assert BHE (Active Low) as low and A0 as high – First byte is transferred through D0-D7 and the second byte is transferred through D8-D15

Example – MOV AX,DS:WORD PTR[0001H]

The memory is made up of semiconductor material used to store the programs and data. Three types of memory is,

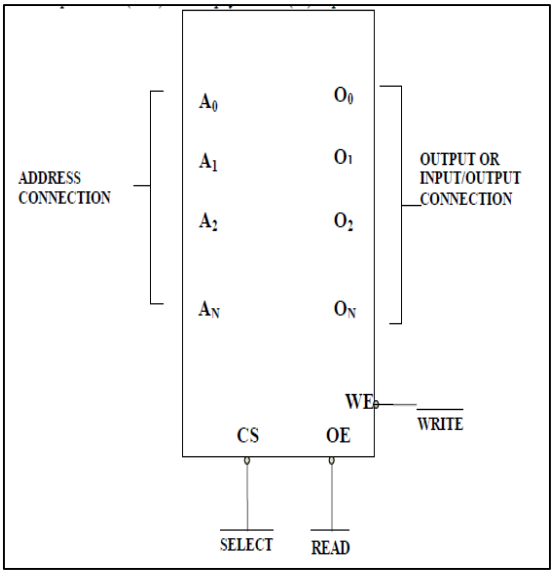
- Process memory
- Primary or main memory
- Secondary memory

Typical EPROM and static RAM:

- A typical semiconductor memory IC will have n address pins, m data pins (or output pins).
- Having two power supply pins (one for connecting required supply voltage (V and the other for connecting ground).
- The control signals needed for static RAM are chip select (chip enable), read control (output enable) and write control (write enable).
- The control signals needed for read operation in EPROM are chip select (chip enable) and read control (output enable).

Pin connections common to all memory devices are: The address input, data output or input/outputs, selection input and control input used to select a read or write operation.

- ***Address connections***: All memory devices have address inputs that select a memory location within the memory device. Address inputs are labeled from A0 to An.
- ***Data connections***: All memory devices have a set of data outputs or input/outputs. Today many of them have bi-directional common I/O pins.
- ***Selection connections***: Each memory device has an input, that selects or enables the memory device. This kind of input is most often called a chip select ($\overline{CS}$), chip enable ($\overline{CE}$) or simply select ($\bar{S}$) input.



Memory component illustrating the address, data and, Control connections

RAM memory generally has at least one $\overline{CS}$ or $\bar{S}$ input and ROM at least one $\overline{CE}$. A RAM memory device has either one or two control inputs. If there is one control input it is often called R/W. This pin selects a read operation or a write operation only if the device is selected by the selection input ($\overline{CS}$). If the RAM has two control inputs, they are usually labeled $\overline{WE}$ or $\bar{W}$ and $\overline{OE}$ or G.

The ROM read only memory permanently stores programs and data and data was always present, even when power is disconnected. It is also called as nonvolatile memory.

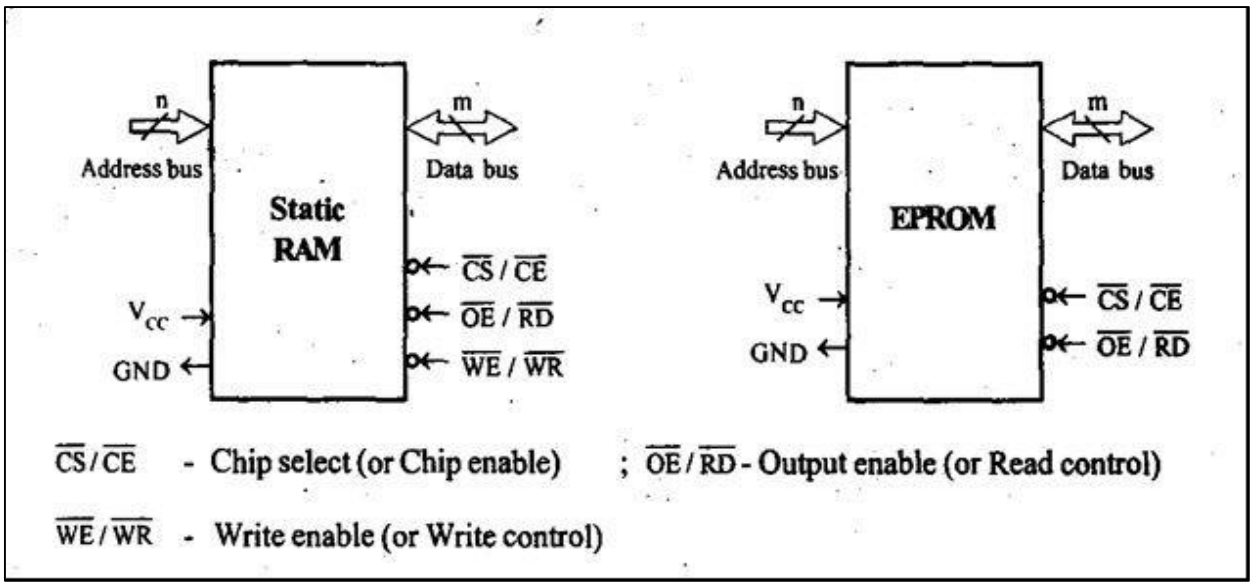
- EPROM (erasable programmable read only memory) is also erasable if exposed to high intensity ultraviolet light for about 20 minutes or less, depending upon the type of EPROM.
- We have PROM (programmable read only memory)
- RMM (read mostly memory) is also called the flash memory.
- The flash memory is also called as an EEPROM (electrically erasable programmable ROM), EAROM (electrically alterable ROM), or a NOVROM (nonvolatile ROM).
- These memory devices are electrically erasable in the system, but require more time to erase than a normal RAM.

- EPROM contains the series of 27XXX contains the following part numbers: 2704(512 * 8), 2708(1K * 8), 2716(2K * 8), 2732(4K * 8), 2764(8K * 8), 27128(16K * 8) etc.
- Each of these parts contains address pins, eight data connections, one or more chip selection inputs ($\overline{CE}$) and an output enable pin ($\overline{OE}$). This device contains **11** address inputs and **8** data outputs. If both the pin connection $\overline{CE}$ and $\overline{OE}$ are at logic 0, data will appear on the output connection. If both the pins are not at logic 0, the data output connections remain at their high impedance or off state.
- To read data from the EPROM Vpp pin must be placed at logic 1.

✓ Static RAM memory device retain data for as long as DC power is applied. Because no special action is required to retain stored data, these devices are called as static memory. They are also called volatile memory because they will not retain data without power.

✓ The main difference between a ROM and RAM is that a RAM is written under normal operation, while ROM is programmed outside the computer and is only normally read.

✓ The SRAM stores temporary data and is used when the size of read/write memory is relatively small.



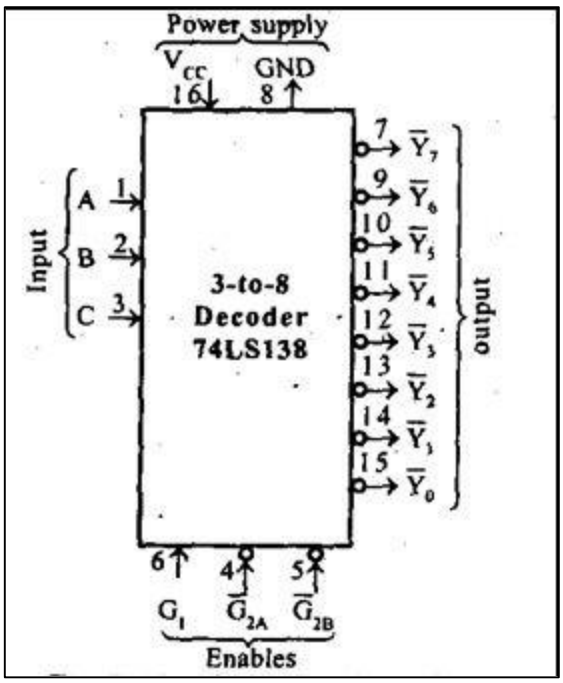
Memory IC EPROM/RAM	Capacity	Number of address pins	Number of data pins
2708/6208	1kb	10	8
2716/6216	2kb	11	8
2732/6232	4kb	12	8
2764/6264	8kb	13	8
27256/62256	32kb	15	8
27512/62512	64kb	16	8
27010/62128	128kb	17	8
27020/62138	256kb	18	8
27040/62148	512kb	19	8

Table - Number of Address Pins and Data Pins in Memory ICs

Decoder:

It is used to select the memory chip of processor during the execution of a program. No of IC's used for decoder is,

- 2-4 decoder (74LS139)
- 3-8 decoder (74LS138)



Input			Output							
C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Fig - Block diagram and Truth table of 3-8 decoder

Static RAM Interfacing

- The semiconductor RAM is broadly two types – Static RAM and Dynamic RAM.
- The semiconductor memories are organized as two dimensional arrays of memory locations.
- For example 4K * 8 or 4K byte memory contains 4096 locations, where each locations contains 8-bit data and only one of the 4096 locations can be selected at a time. Once a location is selected all the bits in it are accessible using a group of conductors called Data bus.
- For addressing the 4K bytes of memory, 12 address lines are required.
- In general to address a memory location out of N memory locations, we will require at least n bits of address, i.e. n address lines where n = Log2 N.
- Thus if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where 2n=N. If out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining (n-p) higher order address lines may be used for address decoding as inputs to the chip selection logic.
- The memory address depends upon the hardware circuit used for decoding the chip select ($\overline{CS}$). The output of the decoding circuit is connected with the $\overline{CS}$ pin of the memory chip.

- The general procedure of static memory interfacing with 8086 is briefly described as follows:
  1. Arrange the available memory chip so as to obtain 16- bit data bus width. The upper 8-bit bank is called as odd address memory bank and the lower 8-bit bank is called as even address memory bank.
  2. Connect available memory address lines of memory chip with those of the microprocessor and also connect the memory $\overline{RD}$ and $\overline{WR}$ 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, $\overline{BHE}$ and A0 are used for decoding the required chip select signals for the odd and even memory banks. The $\overline{CS}$ of memory is derived from the o/p of the decoding circuit.

- As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should not be no windows in the map and no fold back space should be allowed.
- A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred and minimum hardware should be used for decoding.

Example: Interface 8K*8 EPROM and 4K*8 SRAM chips with 8086 microprocessor. Select suitable map.

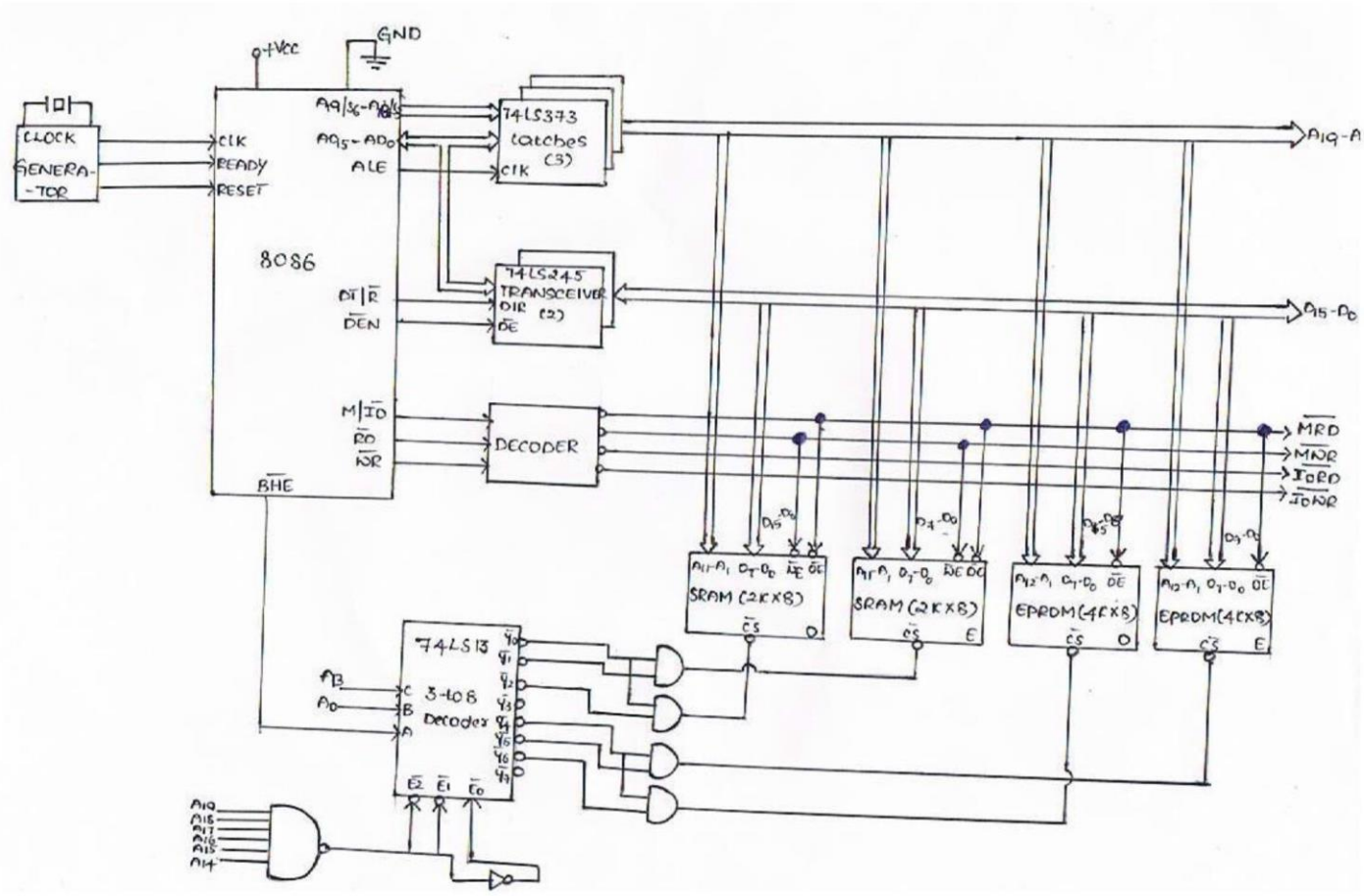
Memory map:

	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
FFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM																				
FE000	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFF	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
SRAM																				
FC000	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Memory chip selection:

Input C	B	A	Data transfer	Memory chips selection
$A13$	$A0$	$\overline{BHE}$		
0	0	0	Word transfer through D15-D0	Both even & odd banks in SRAM
0	0	1	Lower byte transfer through D7-D0	Even bank in SRAM
0	1	0	Higher byte through D15-D8	Odd bank in SRAM
1	0	0	Word transfer through D15-D0	Both even & odd banks in EPROM
1	0	1	Lower byte transfer through D7-D0	Even bank in EPROM
1	1	0	Higher byte through D15-D8	Odd bank in EPROM

Interfacing:



DIRECT MEMORY ACCESS

Need for DMA & DMA Transfer method

An important aspect governing the Computer System performance is the transfer of data between memory and I/O devices. The operation involves loading programs or data files from disk into memory, saving file on disk, and accessing virtual memory pages on any secondary storage medium. Consider a typical system consisting of a CPU, memory and one or more input/output devices as shown in fig. Assume one of the I/O devices is a disk drive and that the computer must load a program from this drive into memory. The CPU would read the first byte of the program and then write that byte to memory. Then it would do the same for the second byte, until it had loaded the entire program into memory. This process proves to be inefficient. Loading data into, and then writing data out of the CPU significantly slows down the transfer. The CPU does not modify the data at all, so it only serves as an additional stop for data on the way to it's final destination. The process would be much quicker if we could bypass the CPU & transfer data directly from the I/O device to memory. Direct Memory Access does exactly that.

A DMA controller implements direct memory access in a computer system.

K SUDHAKAR Unit-3

It connects directly to the I/O device at one end and to the system buses at the other end. It also interacts with the CPU, both via the system buses and two new direct connections. It is sometimes referred to as a channel. In an alternate configuration, the DMA controller may be incorporated directly into the I/O device.

Direct Memory Access--the ability of an I/O subsystem to transfer data to and from a memory subsystem without processor intervention

DMA Controller--a device that can control data transfers between an I/O subsystem and a memory subsystem in the same manner that a processor can control such transfers.

The DMA controller can issue commands to the memory that behave exactly like the commands issued by the microprocessor. The DMA controller in a sense is a second processor in the system but is dedicated to an I/O function. The DMA controller as shown below connects one or more I/O ports directly to memory, where the I/O data stream passes through the DMA controller faster and more efficiently than through the processor as the DMA channel is specialised to the data transfer task.

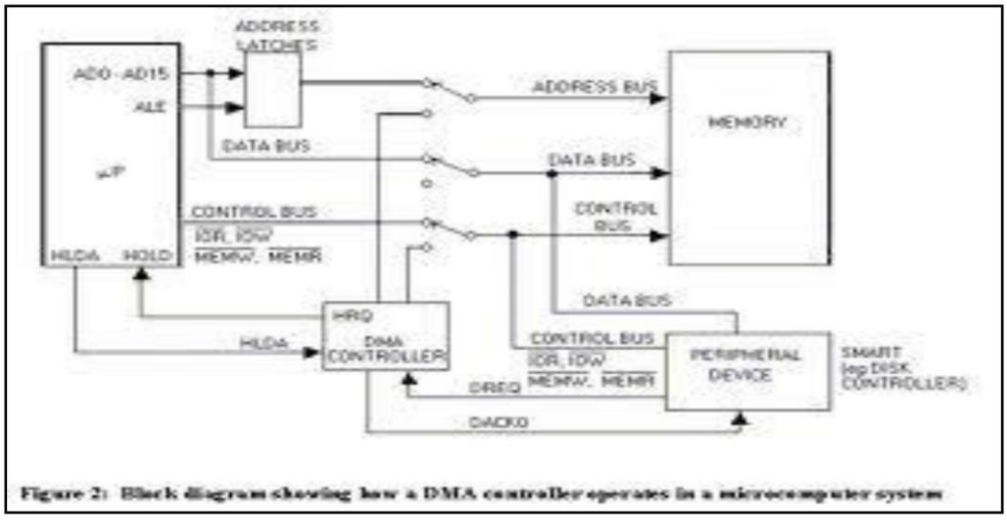


Figure 2: Block diagram showing how a DMA controller operates in a microcomputer system

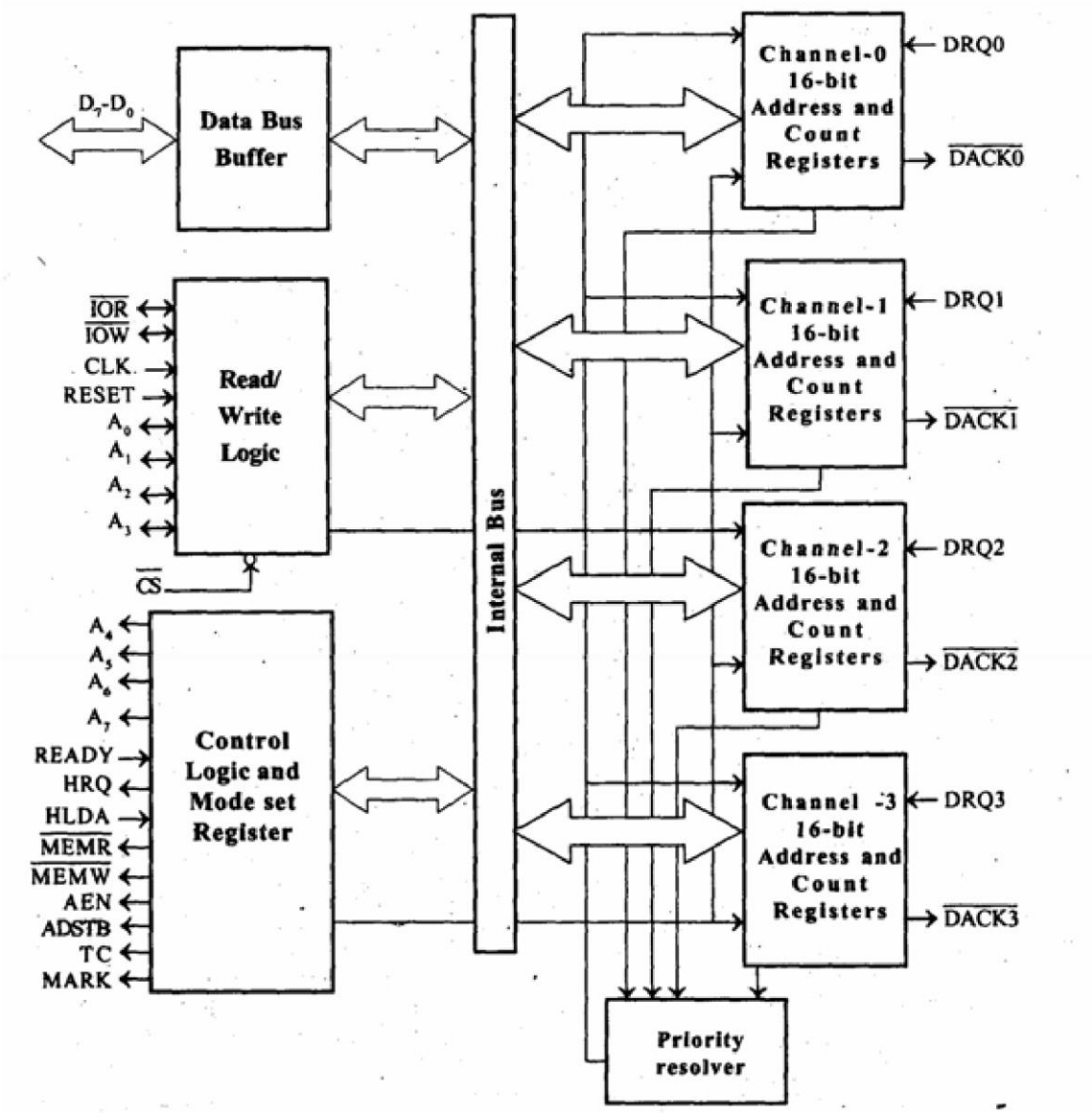
8257 DMA CONTROLLER

The Intel 8257 is a 4-channel direct memory access (DMA) controller. It is specifically designed to simplify the transfer of data at high speeds for the Intel® microcomputer systems. Its primary function is to generate, upon a peripheral request, a sequential memory address which will allow the peripheral to read or write data directly to or from memory. The 8257 has priority logic that resolves the peripherals requests and issues a composite hold request to the microprocessor. It maintains the DMA cycle count for each channel and outputs a control signal to notify the peripheral that the programmed number of DMA cycles is complete.

Features:

- Compatible with 8085, 8086/88
- It is a 4-Channel DMA Controller. So 4- I/O devices can be interfaced to DMA.
- Each channel has 16-bit address and 14 bit counter.
- It provides chip priority resolver that resolves priority of channels in fixed or rotating mode.
- Provides Terminal Count and Modulo 128 Outputs
- It requires Single TTL Clock
- It requires Single + 5V power Supply
- Available in Standard Temperature Range

Architecture:



Block Diagram Description

DMA Channels

The 8257 provides four separate DMA channels (labelled CH-0 to CH-3). Each channel includes two sixteen-bit registers: (1) a DMA address register, and (2) a terminal count register. Both registers must be initialized before a channel is enabled. The DMA address register is loaded with the address of the first memory location to be accessed. The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output to be active in the first DMA cycle for that channel. In general, if N = the number of desired DMA cycles, load the value N-1 into the low-order 14-bits of the terminal count register. The most significant two bits of the terminal count register specify the type of DMA operation for that channel.

DMA Request (DRQ 0-DRQ 3): These are individual asynchronous channel request inputs used by the peripherals to obtain a DMA cycle. If not in the rotating priority mode then DRQ 0 has the highest priority and DRQ 3 has the lowest. A request can be generated by raising the request line and holding it high until DMA acknowledge. For multiple DMA cycles (Burst Mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

DMA Acknowledge (DACK 0 - DACK 3) : An active low level on the acknowledge output informs the peripheral connected to that channel that it has been selected for a DMA cycle. The DACK output acts as a "chip select' for the peripheral device requesting service. This line goes active (low) and inactive (high) once for each byte transferred even if a burst of data is being transferred.

Data bus buffer

This bi-directional 8-bit interfaces the 8257 to the microprocessor system data bus.

Data Bus Lines: These are bi-directional three-state lines. When the 8257 is being programmed by the CPU. Eightbits of data for a DMA address register, a terminal count register or the Mode Set register are received on the data bus. When the CPU reads a DMA address register, a terminal count register or the Status register, the data is sent to the CPU over the data bus. During DMA cycles (when the 8257 is the bus master), the 8257 will output the most significant eight-bits of the memory address (from one of the DMA address registers) to the 8212 latch via the data bus.

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)

Read/Write Logic

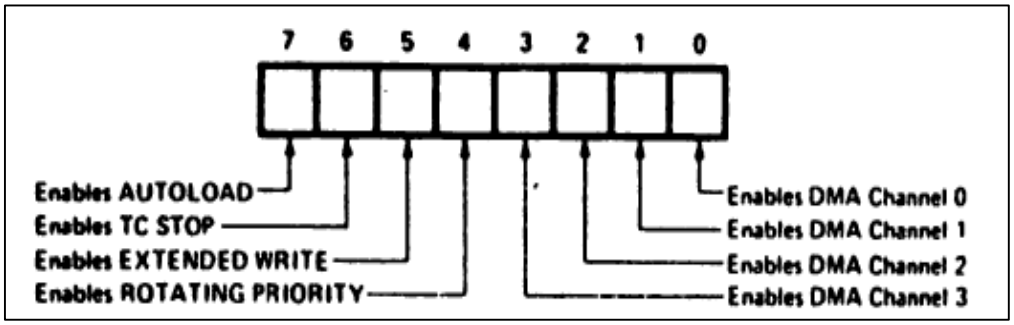
When the CPU is programming or reading one of the 8257's registers (i.e., when the 8257 is a "slave" device on the system bus), the Read/Write Logic accepts the I/O Read (USE) or I/O Write (175OT) signal, decodes the least significant four address bits, (A0-A3), and either writes the contents of the data bus into the addressed register (if I/OW is true) or places the contents of the addressed register onto the data bus (if I/OR is true). During DMA cycles (i.e., when the 8257 is the bus "master"), the Read/Write Logic generates the I/O read and memory write (DMA write cycle) or I/O Write and memory read (DMA read cycle) signals which control the data link with the peripheral that has been granted the DMA cycle.

Control Logic

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed.

Mode Sat Register

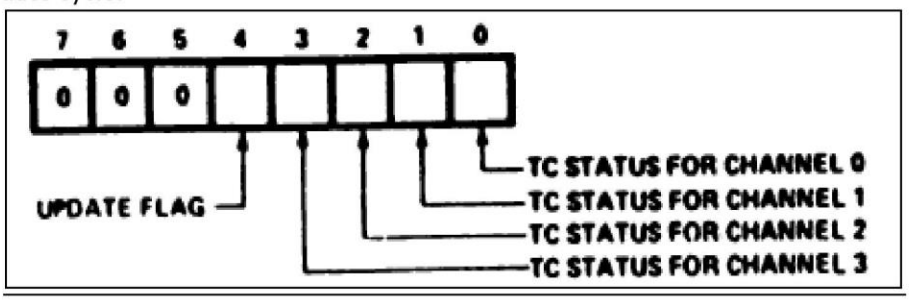
When set, the various bits in the Mode Set register enable each of the four DMA channels, and allow four different options for the 8257:



Status Register

The eight-bit status register indicates which channels have reached a terminal count condition and includes the update flag described previously. The TC status bits are set when the Terminal Count (TC) output is activated for that channel. These bits remain set until the status register is read or the 8257 is reset. The UPDATE FLAG, however, is not affected by

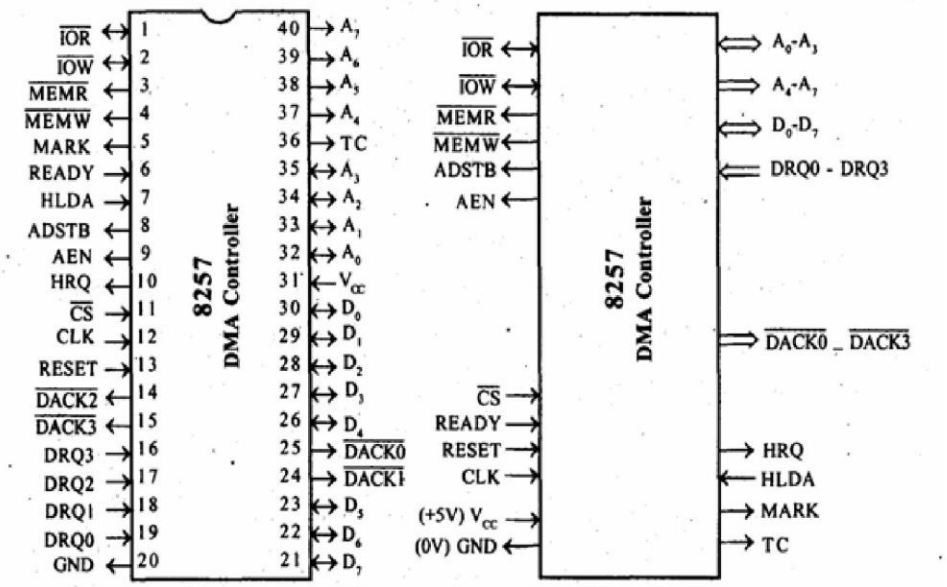
a status register read operation. The UPDATE FLAG can be cleared by resetting the 8257. by changing to the non-auto load mode (i.e.. by resetting the AUTO LOAD bit in the Mode Set register) or it can be left to clear itself at the completion of the update cycle.



Register Selection in 8257

Register	Binary Address								Hexa Address
	Decoder input and enable				Input to address pins of 8257				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Channel-0 DMA address register	0	1	1	0	0	0	0	0	60
Channel-0 count register	0	1	1	0	0	0	0	1	61
Channel-1 DMA address register	0	1	1	0	0	0	1	0	62
Channel-1 count register	0	1	1	0	0	0	1	1	63
Channel-2 DMA address register	0	1	1	0	0	1	0	0	64
Channel-2 count register	0	1	1	0	0	1	0	1	65
Channel-3 DMA address register	0	1	1	0	0	1	1	0	66
Channel-3 count register	0	1	1	0	0	1	1	1	67
Mode set register (Write only)	0	1	1	0	1	0	0	0	68
Status register (Read only)	0	1	1	0	1	0	0	0	68

PIN CONFIGURATION OF 8257



K SUDHAKAR Unit-3

Pin descriptions

D0-D7:

- These are bidirectional, tri state, Buffered, Multiplexed data (D0-D7) and (A8-A15).
- In the slave mode it is a bidirectional (Data is moving).
- In the Master mode it is a unidirectional (Address is moving)

IOR:

- It is active low ,tristate ,buffered ,Bidirectional lines.
- In the slave mode it function as a input line. IOR signal is generated by microprocessor to read the contents 8257 registers.
- In the master mode it function as a output line. IOR signal is generated by 8257 during write cycle.

IOW:

- It is active low ,tristate ,buffered ,Bidirectional control lines.
- In the slave mode it function as a input line. IOR signal is generated by microprocessor to write the contents 8257 registers.
- In the master mode it function as a output line. IOR signal is generated by 8257 during read cycle.

CLK:

- It is the input line ,connected with TTL clock generator.
- This signal is ignored in slave mode.

RESET: Used to clear mode set registers and status registers

A0-A3: These are the tristate, buffer, bidirectional address lines. In slave mode, these lines are used as address inputs lines and internally decoded to access the internal registers. In master mode, these lines are used as address outputs lines,A0-A3 bits of memory address on the lines.

CS: It is active low, Chip select input line. In the slave mode, it is used to select the chip. In the master mode, it is ignored.

A4-A7: These are the tristate, buffer, output address lines. In slave mode, these lines are used as address outputs lines. In master mode, these lines are used as address outputs lines,A0-A3 bits of memory address on the lines.

READY: It is a asynchronous input line. In master mode, When ready is high it is received the signal. When ready is low, it adds wait state between S1 and S3 In slave mode, this signal is ignored.

HRQ: It is used to receiving the hold request signal from the output device.

HLDA: It is acknowledgment signal from microprocessor.

$\overline{MEMR}$: It is active low ,tristate ,Buffered control output line.In slave mode, it is tristated. In master mode, it activated during DMA read cycle.

$\overline{MEMW}$: It is active low ,tristate ,Buffered control input line. In slave mode, it is tristated. In master mode ,it activated during DMA write cycle.

AEN (Address enable): It is a control output line. In master mode ,it is high. In slave mode ,it is low.Used it isolate the system address ,data ,and control lines.

ADSTB (Address Strobe): It is a control output line. Used to split data and address line. It is working in master mode only. In slave mode it is ignore.

TC (Terminal Count): It is a status of output line. It is activated in master mode only. It is high , it selected the peripheral. It is low ,it free and looking for a new peripheral.

MARK:

- It is a modulo 128 MARK output line.
- It is activated in master mode only.
- It goes high, after transferring every 128 bytes of data block.

DRQ0-DRQ3(DMA Request):

- These are the asynchronous peripheral request input signal.
- The request signals are generated by external peripheral device.

$\overline{DACK}$0-$\overline{DACK3}$:

- These are the active low DMA acknowledge output lines.
- Low level indicates that, peripheral is selected for giving the information (DMA cycle).
- In master mode it is used for chip select.

OPERATING MODES OF 8257

The operating modes of 8257 DMA controller are

- Fixed priority mode
- Rotating priority mode
- Extended write mode
- TC stop mode
- Auto load mode

Fixed priority mode: If the ROTATING PRIORITY bit is not set (set to a zero), each DMA channel has a fixed priority. In this mode Channel 0 has the highest priority and Channel 3 has the lowest priority.

***Rotating priority mode*:** In the Rotating Priority Mode, the priority of the channels has a circular sequence. After each DMA cycle, the priority of each channel changes. The channel which had just been serviced will have the lowest priority.

Extended write mode: If the EXTENDED WRITE bit is set, the duration of the MEMW and I/OW signals is extended by activating them earlier in the DMA cycle. Data transfers within micro computer systems proceed asynchronously to allow use of various types of memory and I/O devices with different access times.

TC stop mode: If the TC STOP bit is set a channel is disabled (i.e.. its enable bit is reset) after the Terminal Count (TC) output goes true, thus automatically preventing further DMA operation on that channel. The enable bit for that channel must be re-programmed to continue or begin another DMA operation. If the TC STOP bit is not set. The occurrence of the TC output has no effect on the channel enable bits.

Auto load mode: The Auto Load mode permits Channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks. Channel 2 registers are initialized as usual for the first data block; Channel 3 registers, however, are used to store the block re-initialization parameters (DMA starting address, terminal count and DMA transfer mode).

INTERFACING OF 8257 WITH 8086

--Refer your NOTE BOOK--

K SUDHAKAR Unit-3

UNIT-III
(ARCHITECTURE OF 8086 & INTERFACING)

Syllabus: Pin diagram of 8086-minimum mode and maximum mode of operation, Timing diagram, memory interfacing to 8086 (static RAM and EPROM). Need for DMA, DMA data transfer method, interfacing with 8237/8257.

INTRODUCTION

This unit explains how to design and implement an 8086 based microcomputer system. To design an 8086 based system, it is necessary to know how to interface the 8086 microprocessor with memory and input and output devices. Due to the mismatch in the speed between the microprocessor and other devices, a set of latches and buffers are required to interface the microprocessor with other devices. In this unit, you will learn about the way in which address/data buses, latches and buffers are used in the process of interfacing. To understand the interfacing principles and concepts it is necessary to learn the various types of bus cycles and bus timings. Overall, this unit makes you to understand how 8086 microprocessor is interfaced with memory and peripherals and how an 8086 based microcomputer system works.

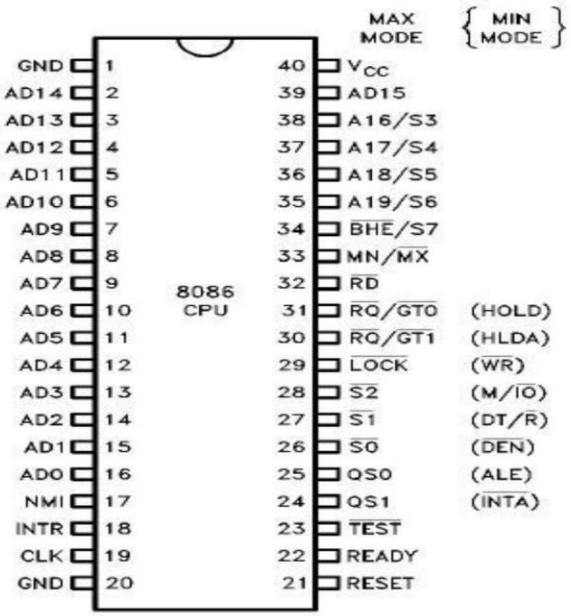
PIN DIAGRAM OF 8086 MICROPROCESSOR

The Microprocessor 8086 is a 16-bit MICROPROCESSOR available in different clock rates (5, 8, 10 MHz) and packaged in a 40 pin DIP or plastic package. The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

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

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

The figure below shows the pins/signals of 8086 processor. Here the pins within the brackets (minimum mode pins) are minimum mode pins.



Signal description:

The 8086 signals can be categorized in three groups.

- The first are the signal having common functions in minimum as well as maximum mode.
- The second are the signals which have special functions for minimum mode.
- Third are the signals which have special functions for maximum mode.

✓ **The following signal descriptions are common for both modes:**

Vcc: It requires +5V single power supply for the operation of the internal circuit.

GND: ground for the internal circuit.

AD15-AD0: These are the time multiplexed memory I/O address and data lines. These lines serve two functions. The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D15 is the MSB and D0 LSB. When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

A19/S6, A18/S5, A17/S4, and A16/S3: These are the time multiplexed address and status lines. During T1 these are the most significant address lines for memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2, T3, Tw and T4.

The status of the interrupt enable flag bit is updated at the beginning of each clock cycle. The status is displayed on S_5 pin.

The **S4** and **S3** combinedly indicate which segment register is presently being used for memory accesses as in below fig.

The last status bit **S6** is always at the logic 0 level.

The address bits are separated from the status bit using latches controlled by the ALE signal.

S_4	S_3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

$\overline{BHE}$**/S7:** The bus high enable 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 transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. $\overline{BHE}$ is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus.

BHE	A_0	Indication
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address
1	1	None

$\overline{RD}$ (Read): This signal on low indicates the peripheral that the processor is performing s memory or I/O read operation. $\overline{RD}$ 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 device or memory that they have completed the data transfer. This signal is provided by an external clock generator device and can be supplied by the memory or I/O subsystem to signal the 8086 when they are ready to permit the data transfer to be completed.

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.

INTR: INTR is an input to the 8086 that can be used by an external device to signal that it needs to be serviced. Logic 1 at INTR represents an active interrupt request. When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the $\overline{INTA}$ output.

$\overline{TEST}$: This input is examined by a 'WAIT' instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state.

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.

RESET: This input causes the processor to terminate the current activity and start execution from FFFF0H.

MN/$\overline{MX}$: The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.

- ✓ **The following pin functions are for the minimum mode operation of 8086:**

M/$\overline{IO}$: This is a status line logically equivalent to S2 in maximum mode. The logic level of M/$\overline{IO}$ tells external circuitry whether a memory or I/O transfer is taking place over the bus. When it is low, it indicates the processor is having an I/O operation, and when it is high, it indicates that the processor is having a memory operation.

$\overline{WR}$: The signal write $\overline{WR}$ indicates that a write bus cycle is in progress. The 8086 switches $\overline{WR}$ to logic 0 to signal external device that valid write or output data are on the bus.

$\overline{INTA}$: (Interrupt Acknowledge)-This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.

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/$\overline{R}$ – Data Transmit/Receive: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

$\overline{DEN}$ **– 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.

HOLD and HLDA: The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals. 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 cycle.

The following pin functions are applicable for maximum mode operation of 8086:
$\overline{S2}$,$\overline{S1}$, **and** $\overline{S0}$ **– Status Lines**: These are the status lines which reflect the type of operation, being carried out by the processor.

Status Inputs			CPU Cycles
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Instruction Fetch
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

LOCK: This output pin indicates that other system bus master will be prevented from gaining the system bus, while the LOCK signal is low.

QS1, QS0 – Queue Status: These lines give information about the status of the code-prefetch queue. Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS0 and QS1. Together they form a 2-bit queue status code, QS1QS0. Following table shows the four different queue status.

QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

RQ0/0GT, RQ1/GT1 – Request/Grant: These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle. Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1.

OPERATING MODES OF 8086

There are two modes of operation for Intel 8086 namely the minimum mode and the maximum mode. When only one 8086 microprocessor is to be used in a micro computer system the 8086 is used in the minimum mode of operation. In this mode the microprocessor issues the control signals required by memory and I/O devices. In a multi processor system it operates in the maximum mode. In case of maximum mode of operation control signals are issued by Intel 8288 bus controller which is used with 8086 for this purpose. The level of the pin $MN/\overline{MX}$ decides the operating mode of 8086. When $MN/\overline{MX}$ is high the microprocessor operates in a minimum mode. When it is low the microprocessor operates in the maximum mode. From pin 24 to 31 issue two different sets of signals. One set of signals is issued when the microprocessor is operating in the minimum mode. The other sort of signal is issued when the microprocessor is operating in the maximum mode. Thus the pins from 24-31 have alternate functions.

TIMINGS:

Timing plays a crucial role, not only in sports like cricket_but also in digital electronic equipments like microprocessors. Timing and Timing diagram plays a vital role in microprocessors. The timing diagram is the diagram which provides information about the various conditions of signals such as high/low, when a machine cycle is being executed. Without the knowledge of timing diagram it is not possible to match the peripheral devices to the microprocessors. These peripheral devices includes memories, ports etc. Such devices can only be matched with microprocessors with the help of timing diagram.

Before dealing with timing diagram, we have to make ourselves familiar with certain terms.

Machine cycle: A basic microprocessor operation such as reading a byte from memory or writing a byte to a port is called a machine cycle (Bus cycle). A machine cycle consists of at least 4 clock cycles/ clock states (T-states) for accessing the data.

Instruction cycle: The time a microprocessor requires to fetch and execute an entire instruction is referred to as an instruction cycle. An instruction cycle consists of one or more machine cycles.

T-state: T-state is nothing but one subdivision of the operation performed in one clock period. These subdivisions are internal state of the microprocessor synchronized with system clock.

So, an instruction cycle is made up of machine cycles, and a machine cycle is made up of states. The time for the state is determined by the frequency of the clock signal.

GENERAL BUS OPERATION CYCLES:

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 transreceivers, when ever 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 above figs shows the signal activities on the 8086 microcomputer buses during simple read and write operations. The first line look at is the clock waveform, CLK, at the top. This represent s the crystal controlled clock signal sent to the 8086 from an external clock generator device such as the 8284. One clock cycle of this clock is called a state. The time interval labeled T1 in the figure is an example of a state. Different versions of the 8086 have maximum clock frequencies of between 5 MHz and 10 MHz, so the minimum time for one sate will be between 100 and 200ns, depending on the part use and the crystal used.

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 $\overline{S0}$, $\overline{S1}$ and $\overline{S2}$ are used to indicate the type of operation. Status bits S3 to S7 are multiplexed with higher order address bits and the $\overline{BHE}$ signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.

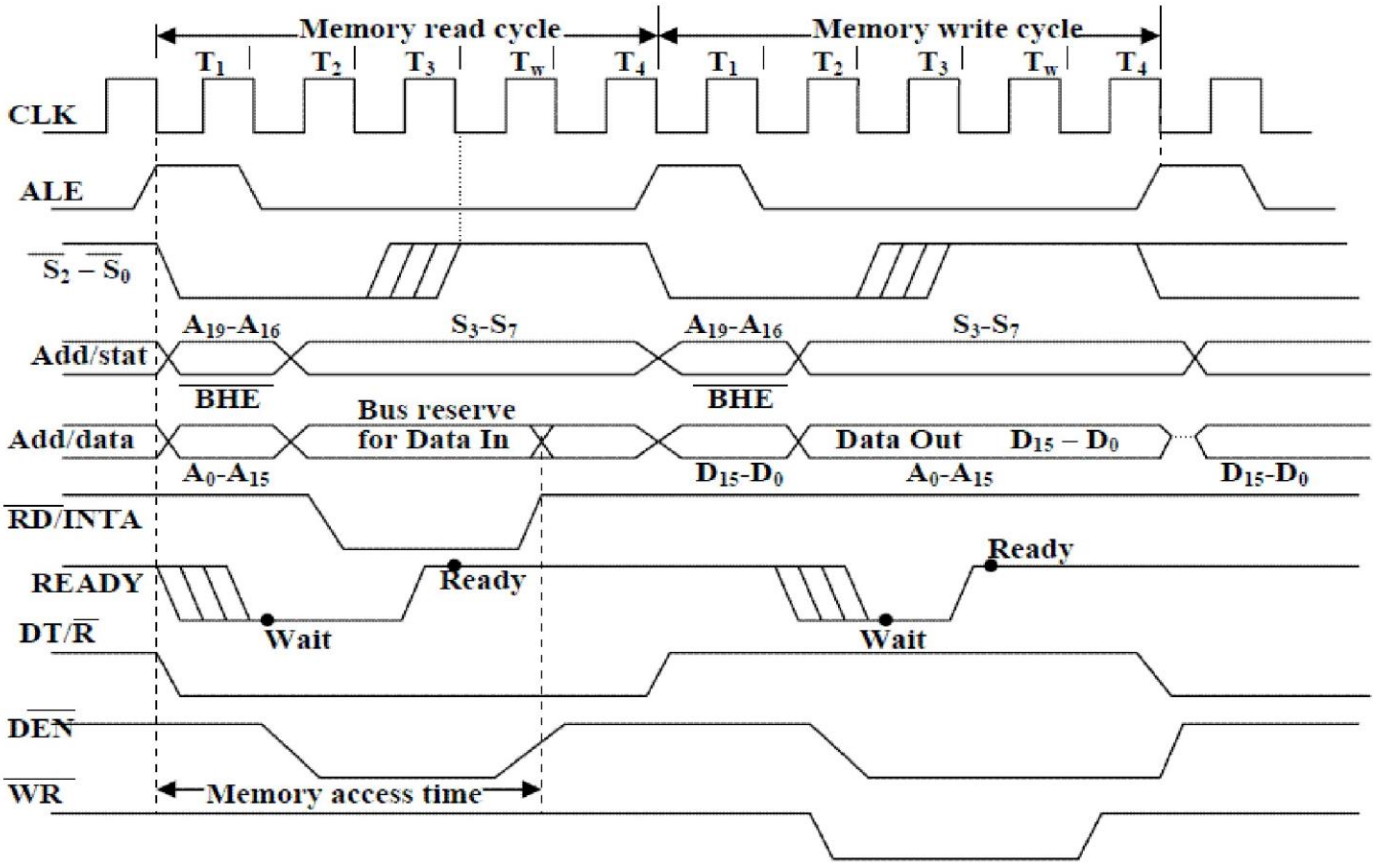


Fig. General bus operation timing diagram

8086 Bus activities during a Read machine cycle:

Fig above (left half portion) shows the timing diagram of 8086 read machine cycle with WAIT state. The clock (CLK) signal is obtained from the clock-generator 8284. Each cycle of the clock is referred to as a state. Minimum number of states to access a data is four. They are T1, T2, T3, and T4 states.

During T1 state of a read machine cycle an 8086 first asserts the M/$\overline{IO}$ signal. It will assert this signal high if it is going to read from memory during memory read cycle and it will assert M/$\overline{IO}$ low if it is going to do a read from an Input port during its read cycle. The timing diagram in fig shows two lines for the M/$\overline{IO}$ signal, because the signal may be going LOW or going HIGH for a read cycle. The point where the two lines cross indicate the time at which the signal becomes valid for this machine cycle.

After asserting M/$\overline{IO}$, the 8086 sends out a high on the address latch enable signal, ALE. The microprocessor sends out on AD0-AD15, A16 through A19 and $\overline{BHE}$ lines, and the address of the memory location that it wants to read. Since the latches are enabled by ALE being high, this address information passes through the latches to their outputs. The 8086 then makes the ALE output low. This disables the latches (8282) and holds the address information latched on the latch outputs. The address information latched on the latch outputs can now be used to select the desired memory or port location.

In the timing diagram, the first point at which the two ($AD_0 - AD_{15}$) cross represents the time at which the 8086 has put a valid address on these lines. Two lines DO NOT indicate that all 16 lines are going high or going low at this point. The crossed lines indicate the time at which a valid address is on the bus.

Since the address information is now held on the latch, the 8086 does not need to send it out any more. As shown in fig. 12 the 8086 floats the AD0 - AD15 lines so that they can be used to input data from memory or from a port. At about the same time the 8086 also remove the $\overline{BHE}$ and A16-A19 information from the upper lines and sends out some status information on these lines.

The 8086 is now ready to read data from the addressed memory locations or port. During T2-state the 8086 asserts its $\overline{RD}$ signal low. This signal is used to enable the addressed memory device or port device.

At the end of T3 state the microprocessor makes the $\overline{RD}$ signal high and reads the data available on the data bus, provided the READY input signal is high. It is the duty of the external circuit to see that valid data is made available on the data bus.

If the READY input pin is not high at the sampled time in a machine cycle, the 8086 will insert one or more WAIT states between T3 and T4 states in that machine cycle. An external hardware device is set up to pulse READY low before the rising edge of the clock in T2 state. After the 8086 finishes T3 of the machine cycle, it enters a WAIT state.

If the READY input is still low at the end of a WAIT state, then the 8086 will insert another WAIT state. The 8086 will continue inserting WAIT states until the READY input is sampled high again. If the READY input is

sampled high again during T3 or during the WAIT state, the microprocessor comes out of the WAIT state and will initiate T4 of the machine cycle.

The $\overline{DEN}$ signal is used to enable bi-directional buffers on the data bus. The data enable signal, $\overline{DEN}$, from the 8086 will enable the data buffer when it is asserted LOW. The data transmit / receive signal DT/$\overline{R}$ from the 8086 is used to specify the direction in which the buffers are enabled. When DT/$\overline{R}$ is asserted high, the buffers will, if enabled by $\overline{DEN}$, transmit data from the 8086 to Memory or I/O ports. When DT/$\overline{R}$ is asserted low, the buffers, if enabled by $\overline{DEN}$, will allow data to be received from Memory or I/O ports of the 8086. DT/$\overline{R}$ is asserted during T1 of the machine cycle. The $\overline{DEN}$ is asserted after the 8086 finishes using the data bus to send the lower 16 address bits

8086 Bus activities during write machine cycle:

The 8086 write operation is very similar to the read cycle. During T1 of a write machine cycle the 8086 asserts M/$\overline{IO}$ low if the write is going to a port and it asserts M/$\overline{IO}$ high if the write is going to memory. At about the same time the 8086 raises ALE high to enable the address latches. The 8086 then assert $\overline{BHE}$ and on the lines AD0 - AD19, it output the address that it will be writing to. When writing to a port, line A16 - A19 will always be low, because the 8086 only sends out 16-bits port addresses. The 8086 brings ALE low again to latch the address on the outputs of the latches. In addition to holding the address, the latches also function as buffers for the address lines. After the address information is latched, the 8086 remove the address information from AD0 - AD15 and outputs the desired data on these lines.

If the READY input is sampled LOW by the 8086 before or during T2 of the machine cycle, the 8086 will insert a WAIT state after T3. If the READY input is sampled high before the end of the WAIT state, the 8086 will go on with state T4 as soon as it completes the WAIT state. The 8086 will continue to insect wait states for as long as the READY is sampled low just before the end of each WAIT state.

MINIMUM MODE 8086 SYSTEM AND TIMINGS

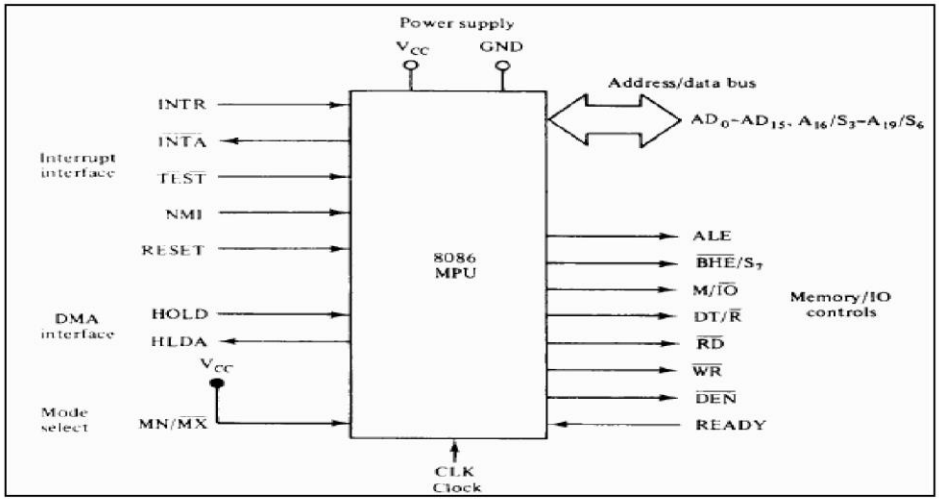


Fig.1: Pin diagram of minimum mode 8086 system

K SUDHAKAR Unit-3

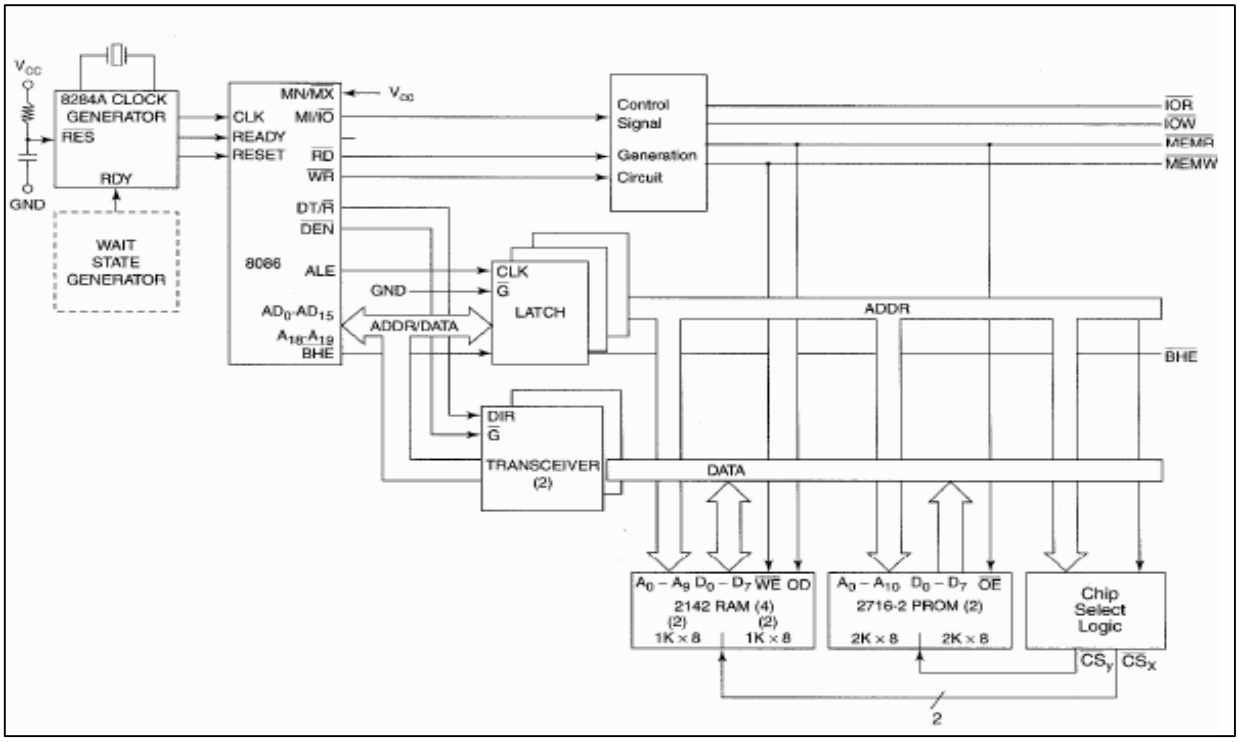


Fig.2: Minimum mode 8086 system configuration

•In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/$\overline{MX}$ pin to logic 1. 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, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

•Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

•Transreceivers are the bidirectional buffers and sometimes they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals. They are controlled by two signals namely, $\overline{DEN}$ and DT/$\bar{R}$. The DT/$\bar{R}$ signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.

•Usually, EPROM is used for monitor storage, while RAM for user's program storage. A system may contain I/O devices.

•The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.

•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.**

Read Cycle:

The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / $\overline{IO}$ signal. During the negative going edge of this signal, the valid address is latched on the local bus.

•The $\overline{BHE}$ and A0 signals address low, high or both bytes. From T1 to T4 , the M/$\overline{IO}$ signal indicates a memory or I/O operation.

•At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read ($\overline{RD}$) control signal is also activated in T2.
•The read ($\overline{RD}$) signal causes the address device to enable its data bus drivers. After $\overline{RD}$ 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.

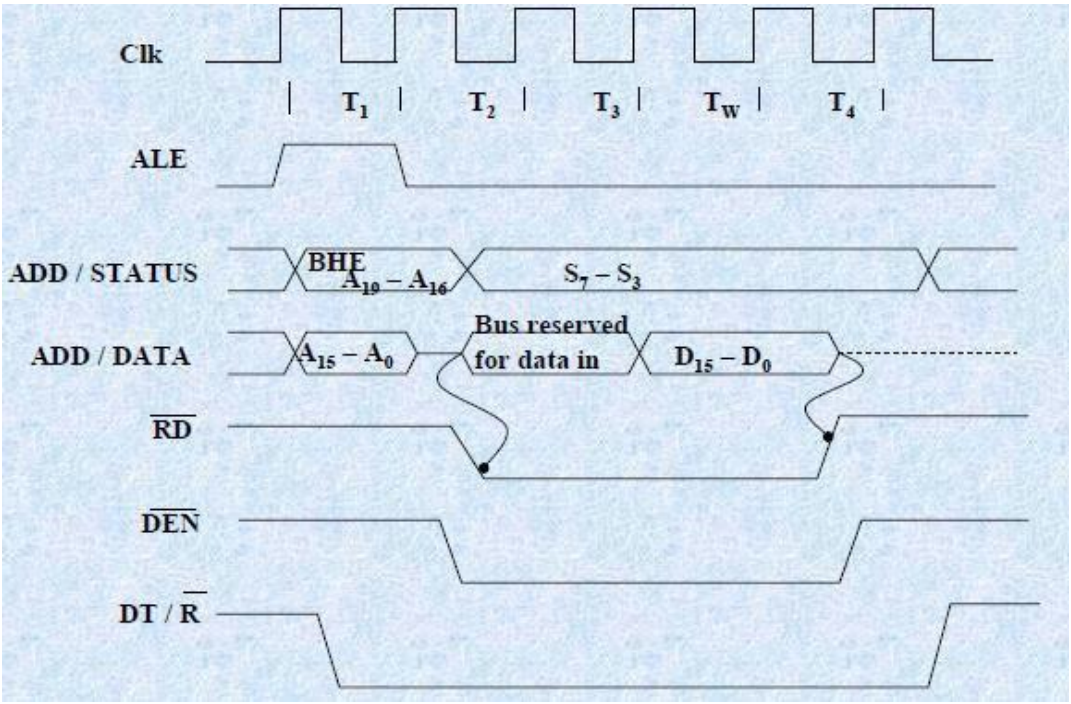


Fig. Read cycle timing diagram for minimum mode operation

Write Cycle:

•A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{IO}$ signal is again asserted to indicate a memory or I/O operation. In T2, after sending the address in T1, the processor sends the data to be written to the addressed location.
•The data remains on the bus until middle of T4 state. The $\overline{WR}$ becomes active at the beginning of T2 (unlike $\overline{RD}$ is somewhat delayed in T2 to provide time for floating).
•The $\overline{BHE}$ and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.

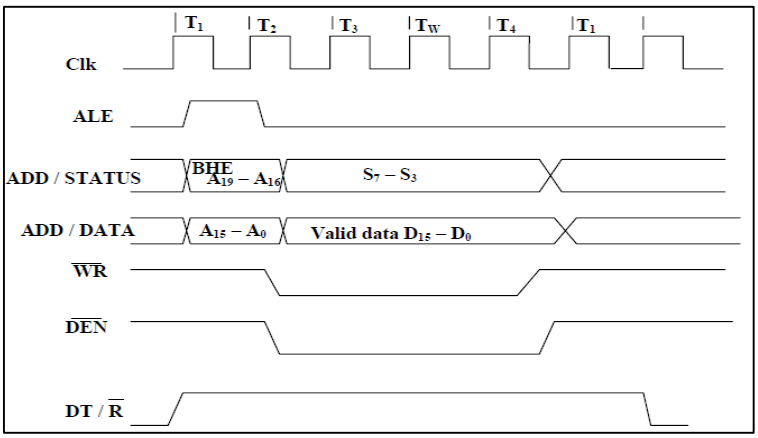


Fig. Write cycle timing diagram for minimum mode operation

The M/$\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ signals indicate the type of data transfer as specified in table below.

M / $\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Transfer Type
0	0	1	I / O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

MAXIMUM MODE 8086 SYSTEM AND TIMINGS

When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment. By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program. Usually in this type of system environment, there are some system resources that are common to all processors. They are called as global resources. There are also other resources that are assigned to specific processors. These are known as local or private resources. Coprocessor also means that there is a second processor in the system. In this two processor does not access the bus at the same time. One passes the control of the system bus to the other and then may suspend its operation. In the maximum-mode 8086 system, facilities are provided for implementing allocation of global resources and passing bus control to other microprocessor or coprocessor.

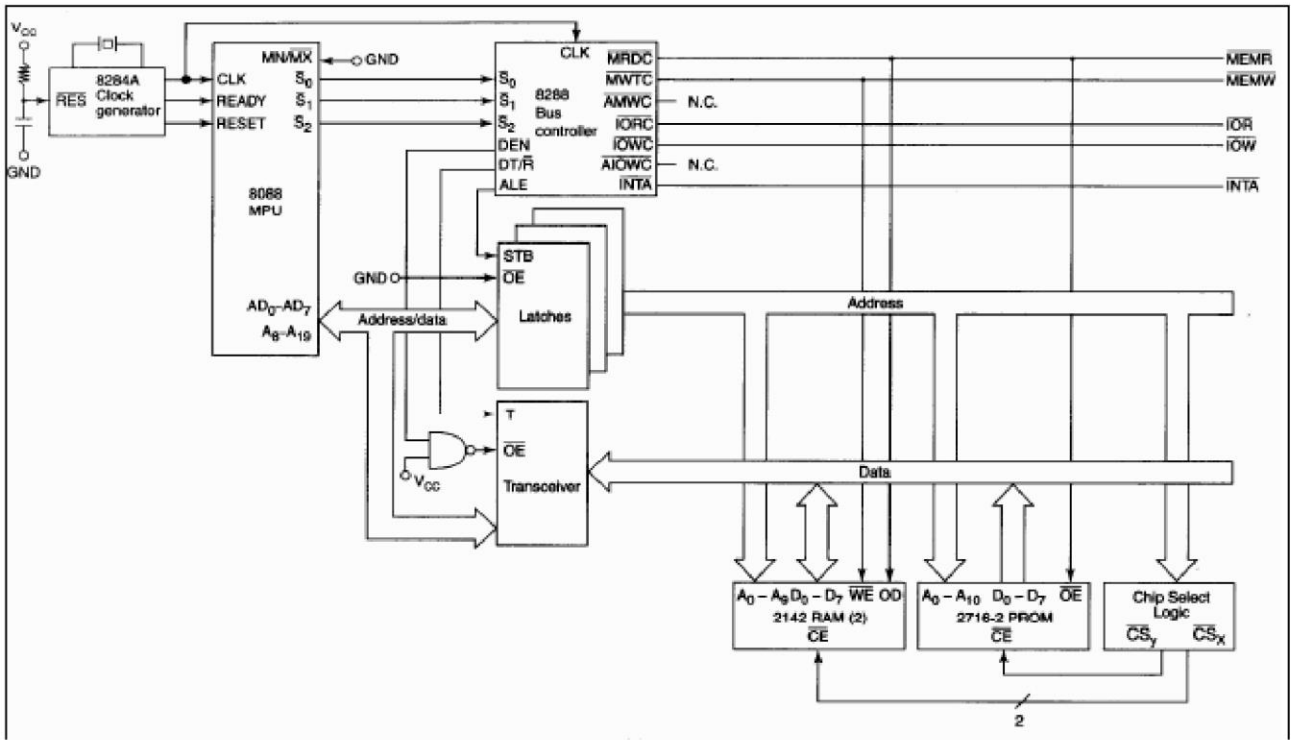


Fig: Maximum mode 8086 system configuration

K SUDHAKAR Unit-3

•In the maximum mode, the 8086 is operated by strapping the MN/$\overline{MX}$ pin to ground. In this mode, the processor derives the status signal $\overline{S2}$, $\overline{S1}$, $\overline{S0}$. Another chip called bus controller derives the control signal using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration. The components in the system are same as in the minimum mode system.

•The basic function of the bus controller chip IC8288 is to derive control signals like $\overline{RD}$ and $\overline{WR}$ (for memory and I/O devices), $\overline{DEN}$, DT/$\overline{R}$, ALE etc. using the information by the processor on the status lines.

•The bus controller chip has input lines $\overline{S2}$, $\overline{S1}$, $\overline{S0}$ and CLK. These inputs to 8288 are driven by MICROPROCESSOR.

•It derives the outputs ALE, $\overline{DEN}$, DT/$\overline{R}$, $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$ and $\overline{AIOWC}$. The AEN, IOB and CEN pins are specially useful for multiprocessor systems.

•AEN and IOB are generally grounded. CEN pin is usually tied to +5V. 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 cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.

•$\overline{INTA}$ pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

•$\overline{IORC}$, $\overline{IOWC}$ 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 address port.

•The $\overline{MRDC}$, $\overline{MWTC}$ are memory read command and memory write command signals respectively and may be used as memory read or write signals.

•All these command signals instructs the memory to accept or send data from or to the bus.

•For both of these write command signals, the advanced signals namely $\overline{AIOWC}$ and $\overline{AMWTC}$ are available.

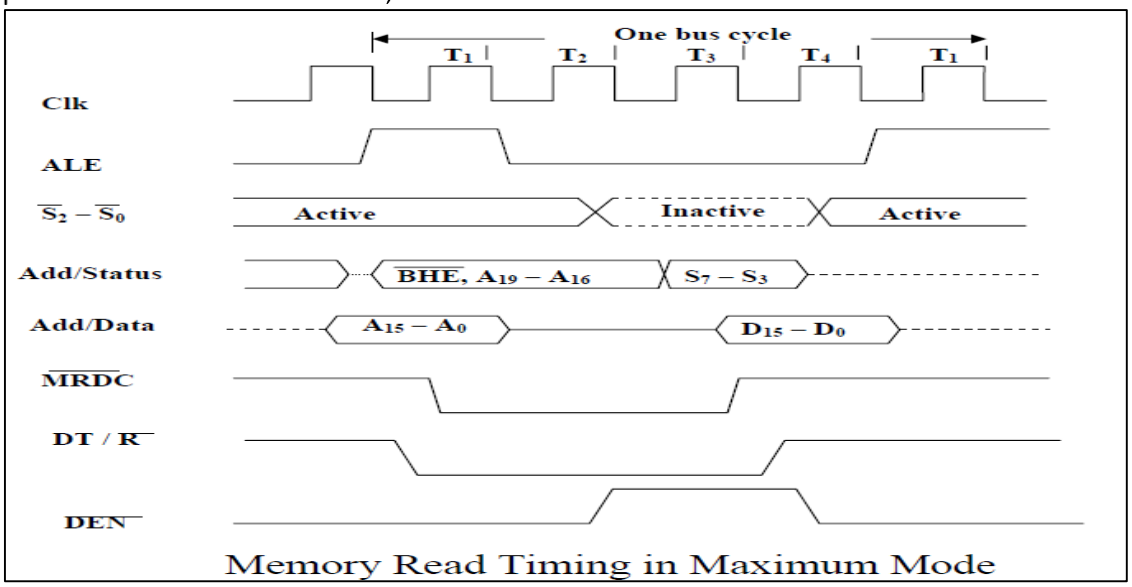
•Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.

•R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.

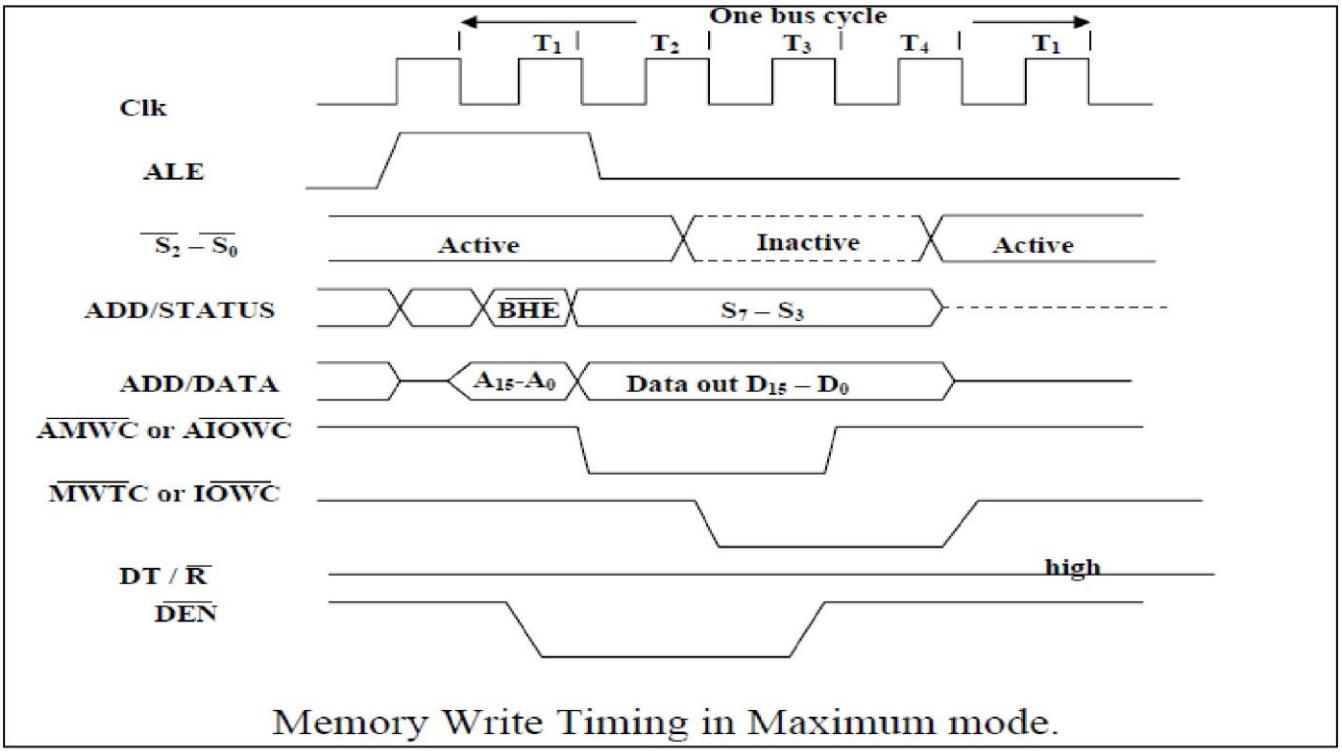
•In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.

•The status bit S0 to S2 remains active until T3 and become passive during T3 and T4.

•If reader input is not activated before T3, wait state will be inserted between T3 and T4.



Memory Read Timing in Maximum Mode



Memory Write Timing in Maximum mode.

ADDRESS AND DATA BUS

The BIU has a combined address and data bus, commonly referred to as a time-multiplexed bus. Time multiplexing address and data information makes the most efficient use of device package pins. A system with address latching provided within the memory and I/O devices can directly connect to the address/data bus. The local bus can be demultiplexed with a single set of address latches to provide non-multiplexed address and data information to the system.

The programmer views the memory or I/O address space as a sequence of bytes. Memory space consists of 1 Mbyte, while I/O space consists of 64 Kbytes. Any byte can contain an 8-bit data element, and any two consecutive bytes can contain a 16-bit data element (identified as a word). The discussions in this section apply to both memory and I/O bus cycles. For brevity, memory bus cycles are used for examples and illustration.

The memory address space on a 16-bit data bus is physically implemented by dividing the address space into two banks of up to 512 Kbytes each (see Figure 1). One bank connects to the lower half of the data bus and contains even-addressed bytes (A0=0). The other bank connects to the upper half of the data bus and contains odd-addressed bytes (A0=1). Address lines A19:1 selects a specific byte within each bank. A0 and Byte High Enable ($\overline{BHE}$) determine whether one bank or both banks participate in the data transfer.

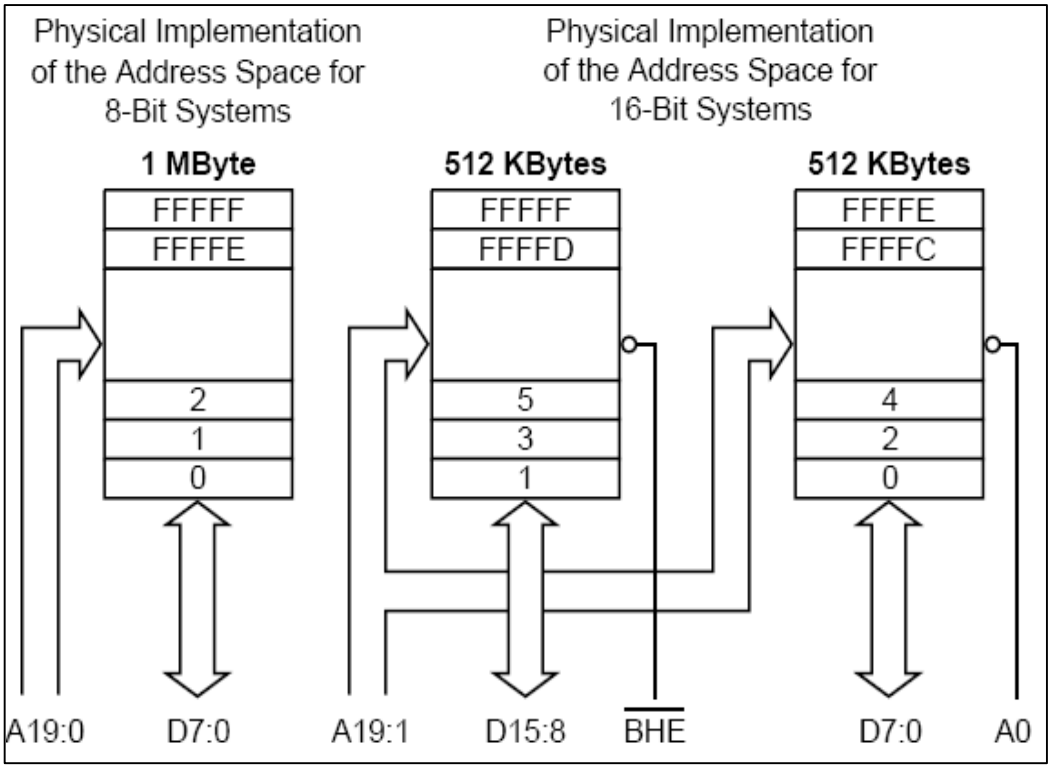


Figure 1. Physical Databus Models

Byte transfers to even addresses transfer information over the lower half of the data bus (see Figure-2a). A0 low enables the lower bank, while BHE high disables the upper bank. The data value from the upper bank is ignored during a bus read cycle. BHE high prevents a write operation from destroying data in the upper bank. Byte transfers to odd addresses transfer information over the upper half of the data bus (see Figure -2b).BHE low enables the upper bank, while A0 high disables the lower bank. The data value from the lower bank is ignored during a bus read cycle. A0 high prevents a write operation from destroying data in the lower bank. To access even-addressed 16-bit words (two consecutive bytes with the least-significant byte at an even address), information is transferred over both halves of the data bus (see Figure-3).

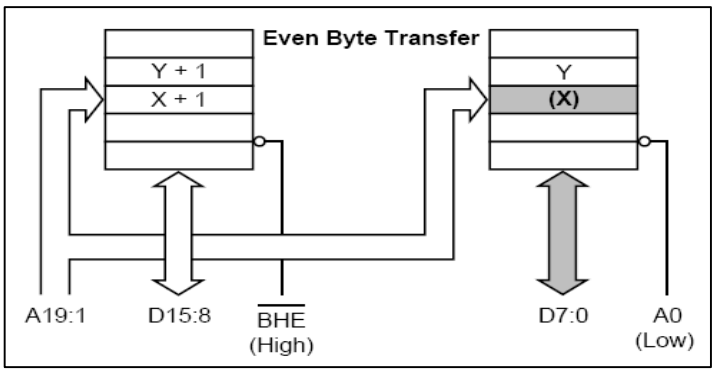


Figure.2a: 16-bit Data Bus Byte Transfers

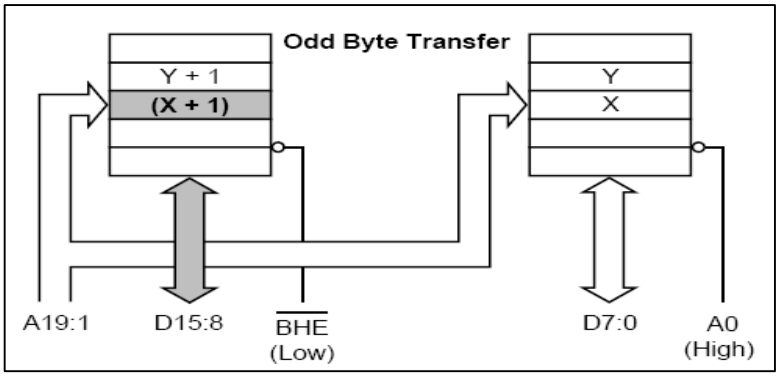


Figure.2b: 16-bit Data Bus Byte transfers

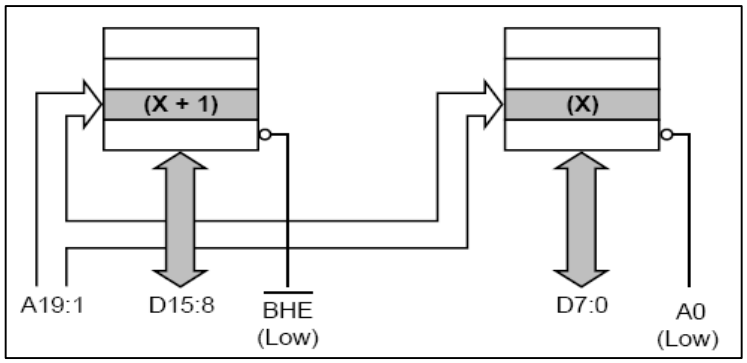


Figure-3: 16-Bit Data Bus Even Word Transfers

A19:1 select the appropriate byte within each bank. A0 and $\overline{BHE}$ drive low to enable both banks simultaneously. Odd-addressed word accesses require the BIU to split the transfer into two byte operations (see Figure-4). The first operation transfers data over the upper half of the bus, while the second operation transfers data over the lower half of the bus. The BIU automatically executes the two-byte sequence whenever an odd-addressed word access is performed.

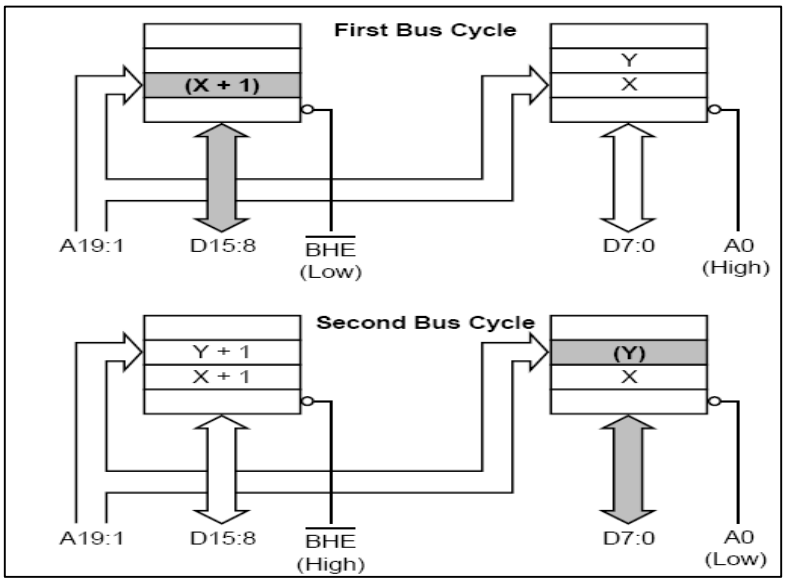


Figure -4 16-Bit data Bus Odd Word Transfers

MEMORY INTERFACING:

8086 Memory Banks

8086 has a 20 bit address bus and hence it can address 2^{20} or 1,048,576 addresses. In each location a byte is stored. So when a word is stored in the memory, it is stored in two consecutive memory locations. Strictly speaking, both memory read and memory write operations require more than one memory cycle. If we want 8086 to complete memory read and memory write operations to be completed with one machine cycle, the memory is to be organized in the form of two banks. Each bank will have 524,288 bytes each.

One memory bank contains all the even addressed locations like 00000, 00002 and 00004. The data lines of this bank are connected to the lower eight data lines, D0 through D7 of the 8086. The other memory bank has all the odd addressed locations like 00001, 00003 and 00005. The data lines of this bank are connected to the upper eight data lines, D8 through D15 of the 8086. Address line A0 is used as part of the enabling for memory in the lower bank. Address lines A1 through A19 are used to select the desired memory device in the bank to address the desired byte in the service. These address lines from A1 through A19 are also used to access a particular location in the upper bank. An additional signal called Bus High Enable (BHE – Active Low) is used to enable the upper memory bank. An external latch, strobed by ALE, grabs the BHE (Active Low) signal that holds it stable for the rest of the machine cycle. The following table shows the required logic levels on the BHE (Active Low) and A0 signals for various types of memory accesses.

Address	Data type	Bhe (active low)	A0	Bus cycles	Data lines used
0000	BYTE	1	0	ONE	D0-D7
0000	WORD	0	0	ONE	D0-D15
0001	BYTE	0	1	ONE	D7-D15
0001	WORD	0 1	1 0	FIRST SECOND	D0-D7 D7-D15

Case 1

Read/Write a byte form/to an even address – A0 will be low and BHE (Active Low) will be high – Byte is transferred to/from low bank through D0-D7

Example – MOV AH, DS: BYTE PTR [0000]

Case 2

Similar to case 1 except the word access instead of the byte access – Both A0 and BHE (Active Low) will be asserted low – Low byte of the word through D0-D7 and high byte of the word through D8-D15

K SUDHAKAR Unit-3

Example – MOV AX,DS:WORD PTR[0000]

Case 3

Read/Wrire a byte from/to an odd address – A0 will be high and BHE (Active Low) will be asserted low – Low bank is disabled and high bank is enabled – Byte is transferred through D0-D7

Example – MOV AL,DS:BYTE PTR[0001]

Case 4

Read/Write a word from/to an odd address – 8086 requires two bus cycles – During the first machine cycle assert BHE (Active Low) as low and A0 as high – First byte is transferred through D0-D7 and the second byte is transferred through D8-D15

Example – MOV AX,DS:WORD PTR[0001H]

The memory is made up of semiconductor material used to store the programs and data. Three types of memory is,

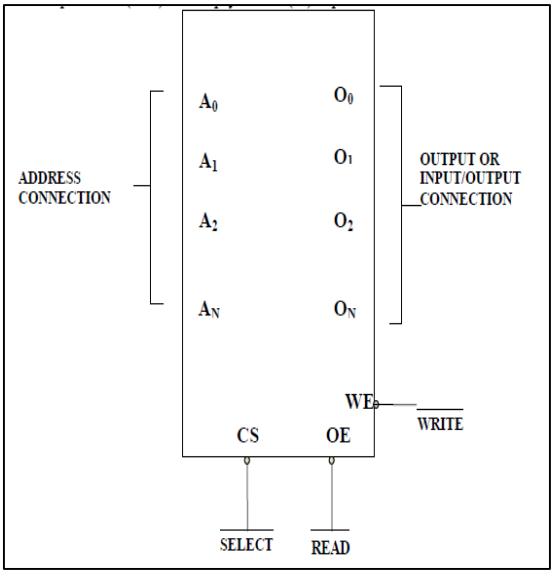
- Process memory
- Primary or main memory
- Secondary memory

Typical EPROM and static RAM:

- A typical semiconductor memory IC will have n address pins, m data pins (or output pins).
- Having two power supply pins (one for connecting required supply voltage (V and the other for connecting ground).
- The control signals needed for static RAM are chip select (chip enable), read control (output enable) and write control (write enable).
- The control signals needed for read operation in EPROM are chip select (chip enable) and read control (output enable).

Pin connections common to all memory devices are: The address input, data output or input/outputs, selection input and control input used to select a read or write operation.

- ***Address connections***: All memory devices have address inputs that select a memory location within the memory device. Address inputs are labeled from A0 to An.
- ***Data connections***: All memory devices have a set of data outputs or input/outputs. Today many of them have bi-directional common I/O pins.
- ***Selection connections***: Each memory device has an input, that selects or enables the memory device. This kind of input is most often called a chip select ($\overline{CS}$), chip enable ($\overline{CE}$) or simply select ($\bar{S}$) input.



Memory component illustrating the address, data and, Control connections

RAM memory generally has at least one $\overline{CS}$ or $\bar{S}$ input and ROM at least one $\overline{CE}$. A RAM memory device has either one or two control inputs. If there is one control input it is often called R/W. This pin selects a read operation or a write operation only if the device is selected by the selection input ($\overline{CS}$). If the RAM has two control inputs, they are usually labeled $\overline{WE}$ or $\bar{W}$ and $\overline{OE}$ or G.

The ROM read only memory permanently stores programs and data and data was always present, even when power is disconnected. It is also called as nonvolatile memory.

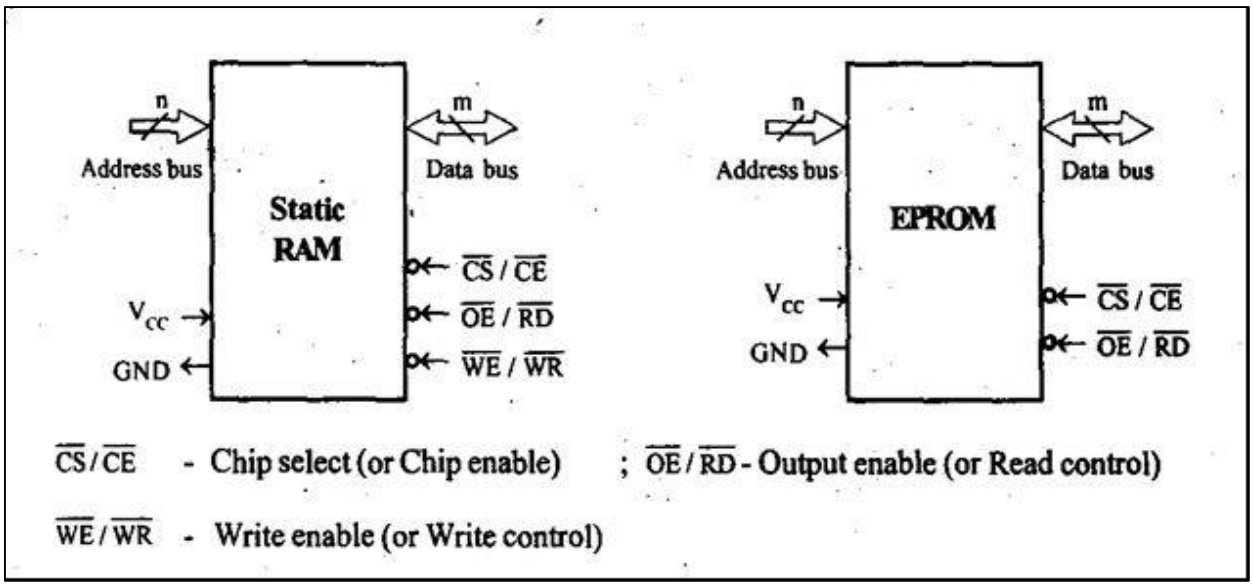
- EPROM (erasable programmable read only memory) is also erasable if exposed to high intensity ultraviolet light for about 20 minutes or less, depending upon the type of EPROM.
- We have PROM (programmable read only memory)
- RMM (read mostly memory) is also called the flash memory.
- The flash memory is also called as an EEPROM (electrically erasable programmable ROM), EAROM (electrically alterable ROM), or a NOVROM (nonvolatile ROM).
- These memory devices are electrically erasable in the system, but require more time to erase than a normal RAM.

- EPROM contains the series of 27XXX contains the following part numbers: 2704(512 * 8), 2708(1K * 8), 2716(2K * 8), 2732(4K * 8), 2764(8K * 8), 27128(16K * 8) etc.
- Each of these parts contains address pins, eight data connections, one or more chip selection inputs ($\overline{CE}$) and an output enable pin ($\overline{OE}$). This device contains **11** address inputs and **8** data outputs. If both the pin connection $\overline{CE}$ and $\overline{OE}$ are at logic 0, data will appear on the output connection. If both the pins are not at logic 0, the data output connections remain at their high impedance or off state.
- To read data from the EPROM Vpp pin must be placed at logic 1.

✓ Static RAM memory device retain data for as long as DC power is applied. Because no special action is required to retain stored data, these devices are called as static memory. They are also called volatile memory because they will not retain data without power.

✓ The main difference between a ROM and RAM is that a RAM is written under normal operation, while ROM is programmed outside the computer and is only normally read.

✓ The SRAM stores temporary data and is used when the size of read/write memory is relatively small.



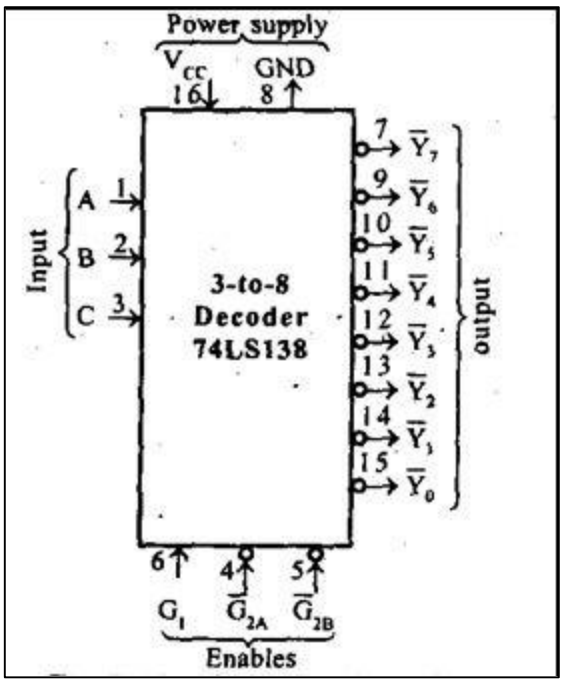
Memory IC EPROM/RAM	Capacity	Number of address pins	Number of data pins
2708/6208	1kb	10	8
2716/6216	2kb	11	8
2732/6232	4kb	12	8
2764/6264	8kb	13	8
27256/62256	32kb	15	8
27512/62512	64kb	16	8
27010/62128	128kb	17	8
27020/62138	256kb	18	8
27040/62148	512kb	19	8

Table - Number of Address Pins and Data Pins in Memory ICs

Decoder:

It is used to select the memory chip of processor during the execution of a program. No of IC's used for decoder is,

- 2-4 decoder (74LS139)
- 3-8 decoder (74LS138)



Input			Output							
C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Fig - Block diagram and Truth table of 3-8 decoder

Static RAM Interfacing

- The semiconductor RAM is broadly two types – Static RAM and Dynamic RAM.
- The semiconductor memories are organized as two dimensional arrays of memory locations.
- For example 4K * 8 or 4K byte memory contains 4096 locations, where each locations contains 8-bit data and only one of the 4096 locations can be selected at a time. Once a location is selected all the bits in it are accessible using a group of conductors called Data bus.
- For addressing the 4K bytes of memory, 12 address lines are required.
- In general to address a memory location out of N memory locations, we will require at least n bits of address, i.e. n address lines where n = Log2 N.
- Thus if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where 2n=N. If out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining (n-p) higher order address lines may be used for address decoding as inputs to the chip selection logic.
- The memory address depends upon the hardware circuit used for decoding the chip select ($\overline{CS}$). The output of the decoding circuit is connected with the $\overline{CS}$ pin of the memory chip.

- The general procedure of static memory interfacing with 8086 is briefly described as follows:
  1. Arrange the available memory chip so as to obtain 16- bit data bus width. The upper 8-bit bank is called as odd address memory bank and the lower 8-bit bank is called as even address memory bank.
  2. Connect available memory address lines of memory chip with those of the microprocessor and also connect the memory $\overline{RD}$ and $\overline{WR}$ 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, $\overline{BHE}$ and A0 are used for decoding the required chip select signals for the odd and even memory banks. The $\overline{CS}$ of memory is derived from the o/p of the decoding circuit.

- As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should not be no windows in the map and no fold back space should be allowed.
- A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred and minimum hardware should be used for decoding.

Example: Interface 8K*8 EPROM and 4K*8 SRAM chips with 8086 microprocessor. Select suitable map.

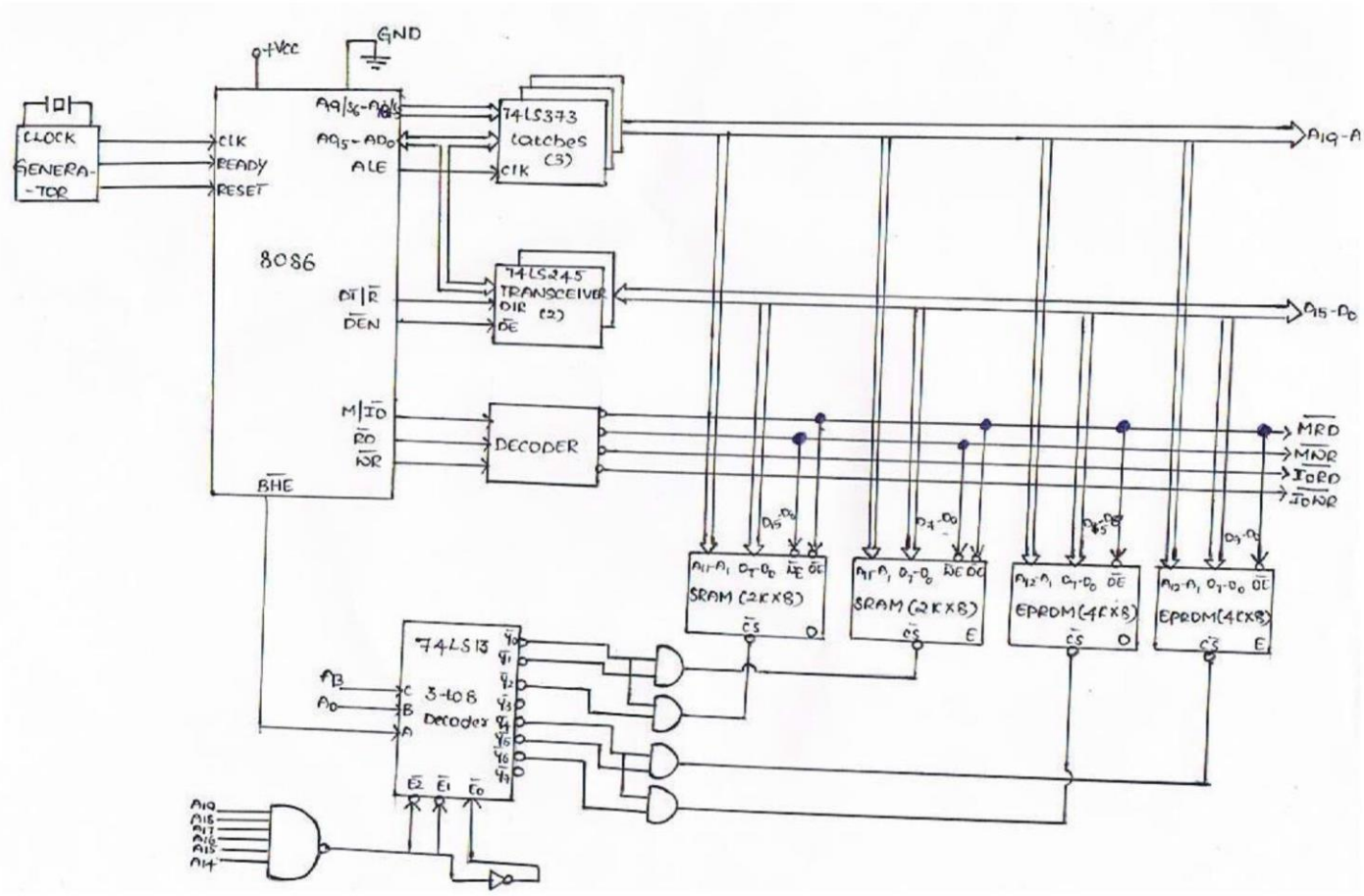
Memory map:

	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
FFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM																				
FE000	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFF	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
SRAM																				
FC000	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Memory chip selection:

Input C	B	A	Data transfer	Memory chips selection
$A13$	$A0$	$\overline{BHE}$		
0	0	0	Word transfer through D15-D0	Both even & odd banks in SRAM
0	0	1	Lower byte transfer through D7-D0	Even bank in SRAM
0	1	0	Higher byte through D15-D8	Odd bank in SRAM
1	0	0	Word transfer through D15-D0	Both even & odd banks in EPROM
1	0	1	Lower byte transfer through D7-D0	Even bank in EPROM
1	1	0	Higher byte through D15-D8	Odd bank in EPROM

Interfacing:



DIRECT MEMORY ACCESS

Need for DMA & DMA Transfer method

An important aspect governing the Computer System performance is the transfer of data between memory and I/O devices. The operation involves loading programs or data files from disk into memory, saving file on disk, and accessing virtual memory pages on any secondary storage medium. Consider a typical system consisting of a CPU, memory and one or more input/output devices as shown in fig. Assume one of the I/O devices is a disk drive and that the computer must load a program from this drive into memory. The CPU would read the first byte of the program and then write that byte to memory. Then it would do the same for the second byte, until it had loaded the entire program into memory. This process proves to be inefficient. Loading data into, and then writing data out of the CPU significantly slows down the transfer. The CPU does not modify the data at all, so it only serves as an additional stop for data on the way to it's final destination. The process would be much quicker if we could bypass the CPU & transfer data directly from the I/O device to memory. Direct Memory Access does exactly that.

A DMA controller implements direct memory access in a computer system.

K SUDHAKAR Unit-3

It connects directly to the I/O device at one end and to the system buses at the other end. It also interacts with the CPU, both via the system buses and two new direct connections. It is sometimes referred to as a channel. In an alternate configuration, the DMA controller may be incorporated directly into the I/O device.

Direct Memory Access--the ability of an I/O subsystem to transfer data to and from a memory subsystem without processor intervention

DMA Controller--a device that can control data transfers between an I/O subsystem and a memory subsystem in the same manner that a processor can control such transfers.

The DMA controller can issue commands to the memory that behave exactly like the commands issued by the microprocessor. The DMA controller in a sense is a second processor in the system but is dedicated to an I/O function. The DMA controller as shown below connects one or more I/O ports directly to memory, where the I/O data stream passes through the DMA controller faster and more efficiently than through the processor as the DMA channel is specialised to the data transfer task.

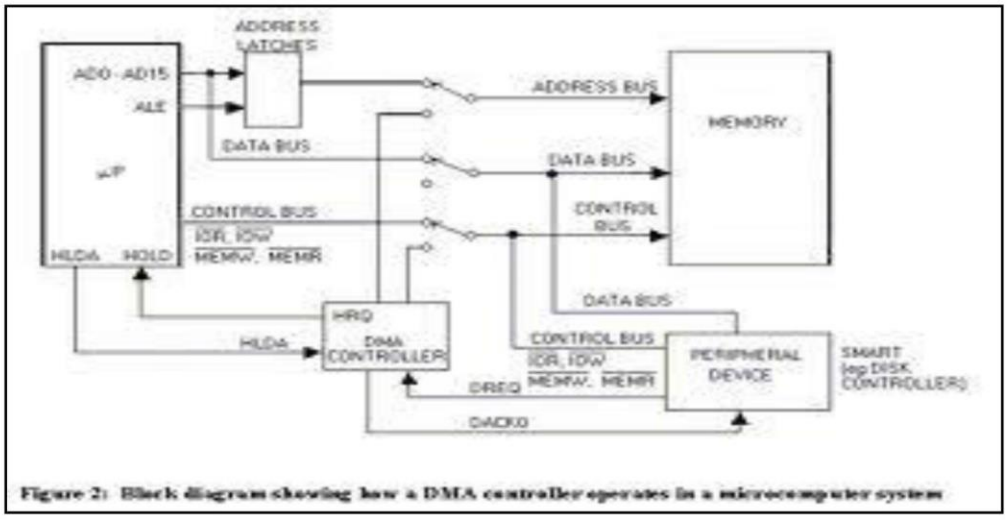


Figure 2: Block diagram showing how a DMA controller operates in a microcomputer system

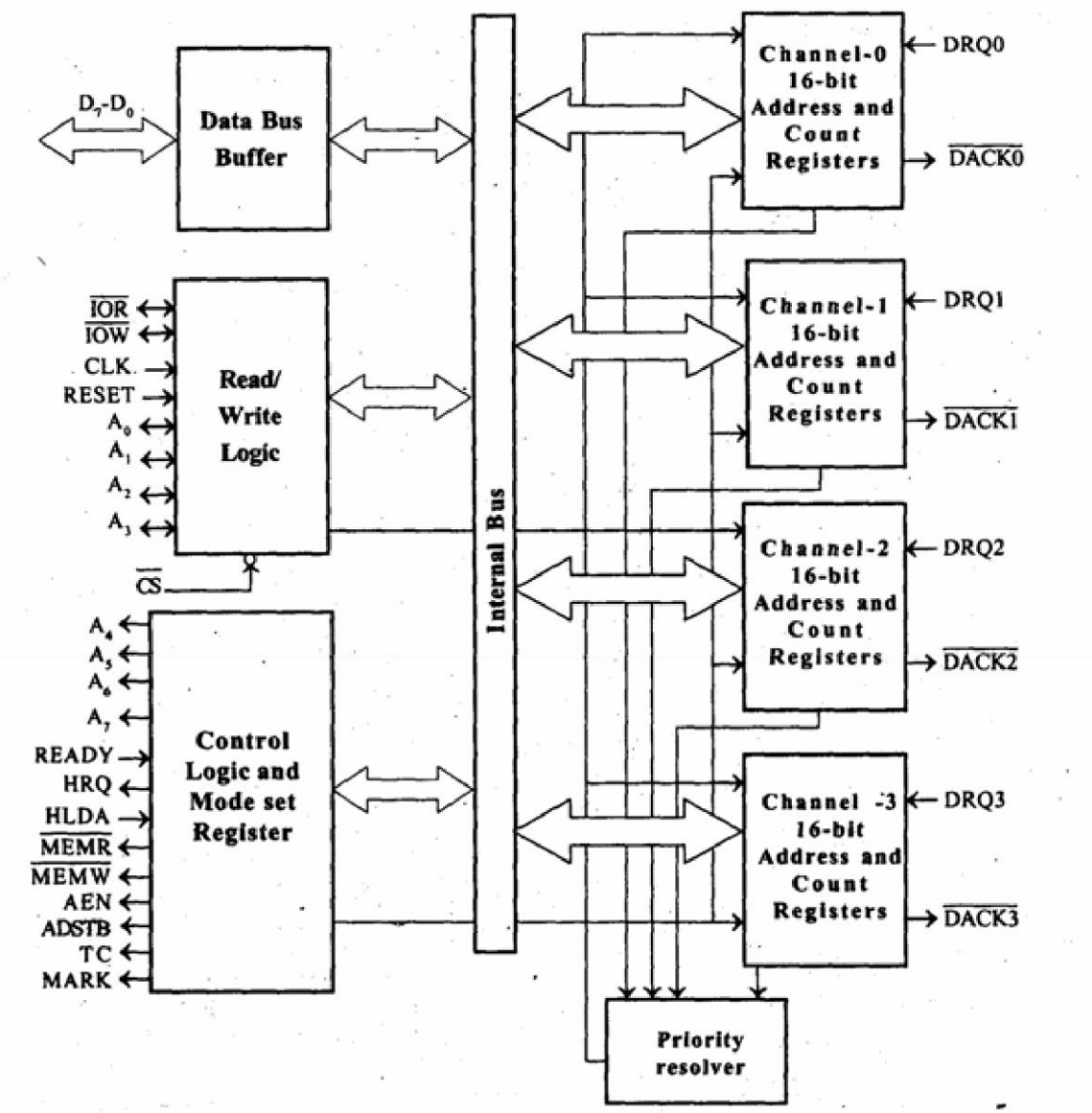
8257 DMA CONTROLLER

The Intel 8257 is a 4-channel direct memory access (DMA) controller. It is specifically designed to simplify the transfer of data at high speeds for the Intel® microcomputer systems. Its primary function is to generate, upon a peripheral request, a sequential memory address which will allow the peripheral to read or write data directly to or from memory. The 8257 has priority logic that resolves the peripherals requests and issues a composite hold request to the microprocessor. It maintains the DMA cycle count for each channel and outputs a control signal to notify the peripheral that the programmed number of DMA cycles is complete.

Features:

- Compatible with 8085, 8086/88
- It is a 4-Channel DMA Controller. So 4- I/O devices can be interfaced to DMA.
- Each channel has 16-bit address and 14 bit counter.
- It provides chip priority resolver that resolves priority of channels in fixed or rotating mode.
- Provides Terminal Count and Modulo 128 Outputs
- It requires Single TTL Clock
- It requires Single + 5V power Supply
- Available in Standard Temperature Range

Architecture:



Block Diagram Description

DMA Channels

The 8257 provides four separate DMA channels (labelled CH-0 to CH-3). Each channel includes two sixteen-bit registers: (1) a DMA address register, and (2) a terminal count register. Both registers must be initialized before a channel is enabled. The DMA address register is loaded with the address of the first memory location to be accessed. The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output to be active in the first DMA cycle for that channel. In general, if N = the number of desired DMA cycles, load the value N-1 into the low-order 14-bits of the terminal count register. The most significant two bits of the terminal count register specify the type of DMA operation for that channel.

DMA Request (DRQ 0-DRQ 3): These are individual asynchronous channel request inputs used by the peripherals to obtain a DMA cycle. If not in the rotating priority mode then DRQ 0 has the highest priority and DRQ 3 has the lowest. A request can be generated by raising the request line and holding it high until DMA acknowledge. For multiple DMA cycles (Burst Mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

DMA Acknowledge (DACK 0 - DACK 3) : An active low level on the acknowledge output informs the peripheral connected to that channel that it has been selected for a DMA cycle. The DACK output acts as a "chip select' for the peripheral device requesting service. This line goes active (low) and inactive (high) once for each byte transferred even if a burst of data is being transferred.

Data bus buffer

This bi-directional 8-bit interfaces the 8257 to the microprocessor system data bus.

Data Bus Lines: These are bi-directional three-state lines. When the 8257 is being programmed by the CPU. Eightbits of data for a DMA address register, a terminal count register or the Mode Set register are received on the data bus. When the CPU reads a DMA address register, a terminal count register or the Status register, the data is sent to the CPU over the data bus. During DMA cycles (when the 8257 is the bus master), the 8257 will output the most significant eight-bits of the memory address (from one of the DMA address registers) to the 8212 latch via the data bus.

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)

Read/Write Logic

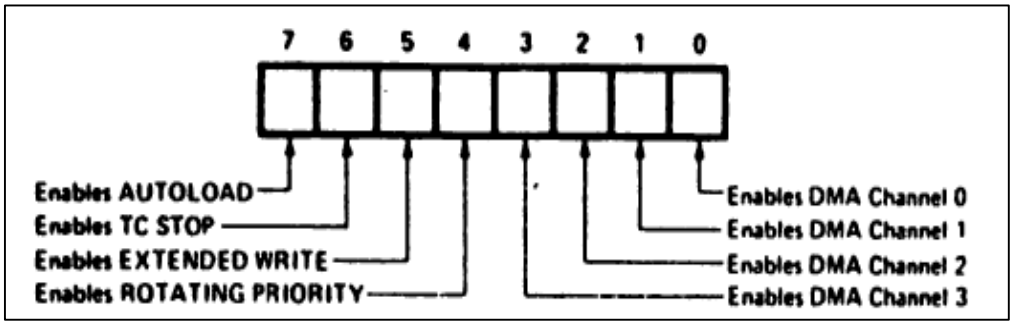
When the CPU is programming or reading one of the 8257's registers (i.e., when the 8257 is a "slave" device on the system bus), the Read/Write Logic accepts the I/O Read (USE) or I/O Write (175OT) signal, decodes the least significant four address bits, (A0-A3), and either writes the contents of the data bus into the addressed register (if I/OW is true) or places the contents of the addressed register onto the data bus (if I/OR is true). During DMA cycles (i.e., when the 8257 is the bus "master"), the Read/Write Logic generates the I/O read and memory write (DMA write cycle) or I/O Write and memory read (DMA read cycle) signals which control the data link with the peripheral that has been granted the DMA cycle.

Control Logic

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed.

Mode Sat Register

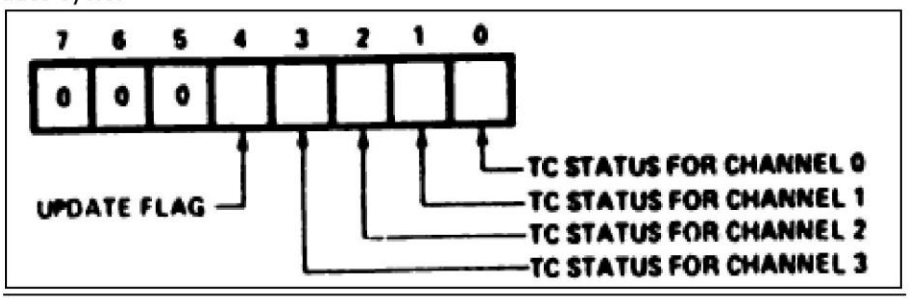
When set, the various bits in the Mode Set register enable each of the four DMA channels, and allow four different options for the 8257:



Status Register

The eight-bit status register indicates which channels have reached a terminal count condition and includes the update flag described previously. The TC status bits are set when the Terminal Count (TC) output is activated for that channel. These bits remain set until the status register is read or the 8257 is reset. The UPDATE FLAG, however, is not affected by

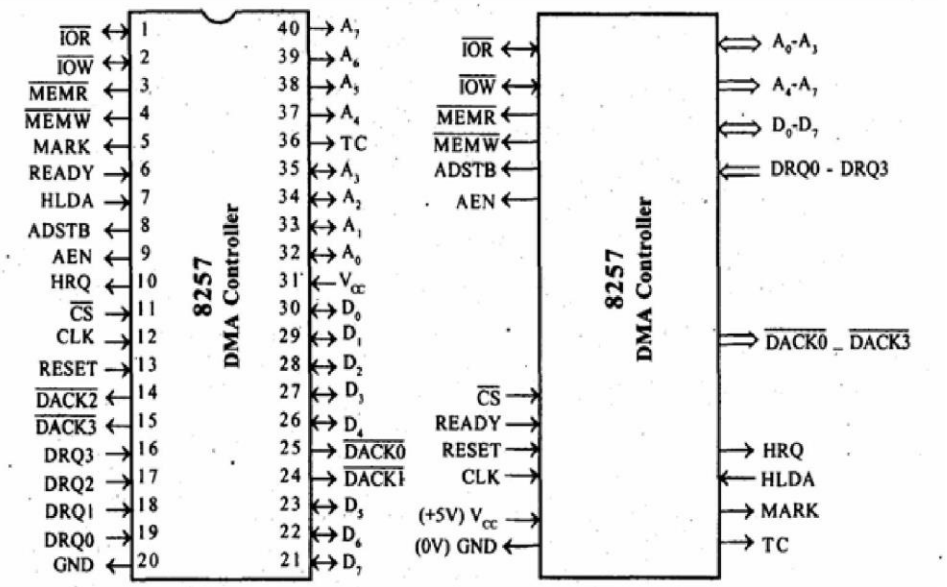
a status register read operation. The UPDATE FLAG can be cleared by resetting the 8257. by changing to the non-auto load mode (i.e.. by resetting the AUTO LOAD bit in the Mode Set register) or it can be left to clear itself at the completion of the update cycle.



Register Selection in 8257

Register	Binary Address								Hexa Address
	Decoder input and enable				Input to address pins of 8257				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Channel-0 DMA address register	0	1	1	0	0	0	0	0	60
Channel-0 count register	0	1	1	0	0	0	0	1	61
Channel-1 DMA address register	0	1	1	0	0	0	1	0	62
Channel-1 count register	0	1	1	0	0	0	1	1	63
Channel-2 DMA address register	0	1	1	0	0	1	0	0	64
Channel-2 count register	0	1	1	0	0	1	0	1	65
Channel-3 DMA address register	0	1	1	0	0	1	1	0	66
Channel-3 count register	0	1	1	0	0	1	1	1	67
Mode set register (Write only)	0	1	1	0	1	0	0	0	68
Status register (Read only)	0	1	1	0	1	0	0	0	68

PIN CONFIGURATION OF 8257



K SUDHAKAR Unit-3

Pin descriptions

D0-D7:

- These are bidirectional, tri state, Buffered, Multiplexed data (D0-D7) and (A8-A15).
- In the slave mode it is a bidirectional (Data is moving).
- In the Master mode it is a unidirectional (Address is moving)

IOR:

- It is active low ,tristate ,buffered ,Bidirectional lines.
- In the slave mode it function as a input line. IOR signal is generated by microprocessor to read the contents 8257 registers.
- In the master mode it function as a output line. IOR signal is generated by 8257 during write cycle.

IOW:

- It is active low ,tristate ,buffered ,Bidirectional control lines.
- In the slave mode it function as a input line. IOR signal is generated by microprocessor to write the contents 8257 registers.
- In the master mode it function as a output line. IOR signal is generated by 8257 during read cycle.

CLK:

- It is the input line ,connected with TTL clock generator.
- This signal is ignored in slave mode.

RESET: Used to clear mode set registers and status registers

A0-A3: These are the tristate, buffer, bidirectional address lines. In slave mode, these lines are used as address inputs lines and internally decoded to access the internal registers. In master mode, these lines are used as address outputs lines,A0-A3 bits of memory address on the lines.

CS: It is active low, Chip select input line. In the slave mode, it is used to select the chip. In the master mode, it is ignored.

A4-A7: These are the tristate, buffer, output address lines. In slave mode, these lines are used as address outputs lines. In master mode, these lines are used as address outputs lines,A0-A3 bits of memory address on the lines.

READY: It is a asynchronous input line. In master mode, When ready is high it is received the signal. When ready is low, it adds wait state between S1 and S3 In slave mode, this signal is ignored.

HRQ: It is used to receiving the hold request signal from the output device.

HLDA: It is acknowledgment signal from microprocessor.

$\overline{MEMR}$: It is active low ,tristate ,Buffered control output line.In slave mode, it is tristated. In master mode, it activated during DMA read cycle.

$\overline{MEMW}$: It is active low ,tristate ,Buffered control input line. In slave mode, it is tristated. In master mode ,it activated during DMA write cycle.

AEN (Address enable): It is a control output line. In master mode ,it is high. In slave mode ,it is low.Used it isolate the system address ,data ,and control lines.

ADSTB (Address Strobe): It is a control output line. Used to split data and address line. It is working in master mode only. In slave mode it is ignore.

TC (Terminal Count): It is a status of output line. It is activated in master mode only. It is high , it selected the peripheral. It is low ,it free and looking for a new peripheral.

MARK:

- It is a modulo 128 MARK output line.
- It is activated in master mode only.
- It goes high, after transferring every 128 bytes of data block.

DRQ0-DRQ3(DMA Request):

- These are the asynchronous peripheral request input signal.
- The request signals are generated by external peripheral device.

$\overline{DACK}$0-$\overline{DACK3}$:

- These are the active low DMA acknowledge output lines.
- Low level indicates that, peripheral is selected for giving the information (DMA cycle).
- In master mode it is used for chip select.

OPERATING MODES OF 8257

The operating modes of 8257 DMA controller are

- Fixed priority mode
- Rotating priority mode
- Extended write mode
- TC stop mode
- Auto load mode

Fixed priority mode: If the ROTATING PRIORITY bit is not set (set to a zero), each DMA channel has a fixed priority. In this mode Channel 0 has the highest priority and Channel 3 has the lowest priority.

***Rotating priority mode*:** In the Rotating Priority Mode, the priority of the channels has a circular sequence. After each DMA cycle, the priority of each channel changes. The channel which had just been serviced will have the lowest priority.

Extended write mode: If the EXTENDED WRITE bit is set, the duration of the MEMW and I/OW signals is extended by activating them earlier in the DMA cycle. Data transfers within micro computer systems proceed asynchronously to allow use of various types of memory and I/O devices with different access times.

TC stop mode: If the TC STOP bit is set a channel is disabled (i.e.. its enable bit is reset) after the Terminal Count (TC) output goes true, thus automatically preventing further DMA operation on that channel. The enable bit for that channel must be re-programmed to continue or begin another DMA operation. If the TC STOP bit is not set. The occurrence of the TC output has no effect on the channel enable bits.

Auto load mode: The Auto Load mode permits Channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks. Channel 2 registers are initialized as usual for the first data block; Channel 3 registers, however, are used to store the block re-initialization parameters (DMA starting address, terminal count and DMA transfer mode).

INTERFACING OF 8257 WITH 8086

--Refer your NOTE BOOK--

K SUDHAKAR Unit-3

NIT -III

8051 MICRO CONTROLLER PROGRAMMING AND APPLICATIONS

3.1 Introduction to micro controllers,

Introduction to Microcontrollers:

8051 is one of the first most popular microcontroller also known as MCS-51. It was introduced by Intel in the year 1981. Initially it came out as N-type metal-oxide-semiconductor (NMOS) based microcontroller, but later versions were based on complementary metal-oxide-semiconductor(CMOS) technology. These microcontrollers were named as 80C51, where C in the name tells that it is based on CMOS technology.

It is an 8-bit microcontroller which means data bus is of 8-bits. Therefore, it can process 8-bits at a time. It is used in wide variety of embedded systems like robotics, remote controls, automotive industry, telecom applications, power tools etc

3. 2 Functional block diagram,.

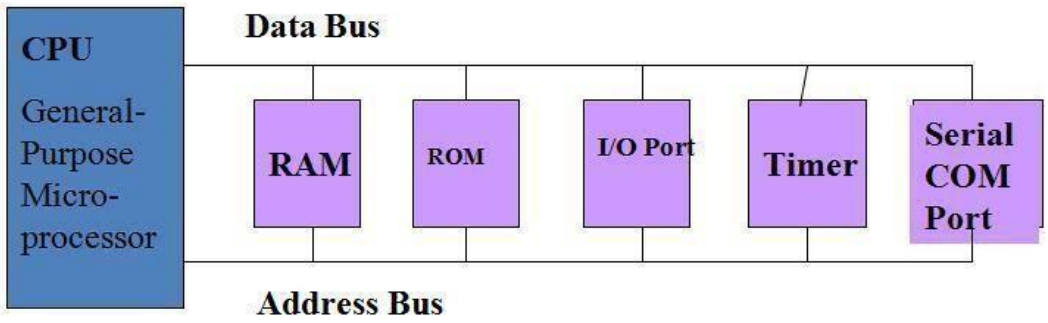
The necessary tools for a microprocessor/controller:

- CPU: Central Processing Unit
- I/O: Input /Output
- Bus: Address bus & Data bus
- Memory: RAM & ROM
- Timer
- Interrupt
- Serial Port
- Parallel Port
-

Microprocessors:

General-purpose microprocessor :

- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example：Intel’s x86, Motorola’s 680x0



Microcontroller :

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example：Motorola’s 6811, Intel’s 8051, Zilog’s Z8 and PIC 16X
-

Microprocessor vs. Microcontroller :

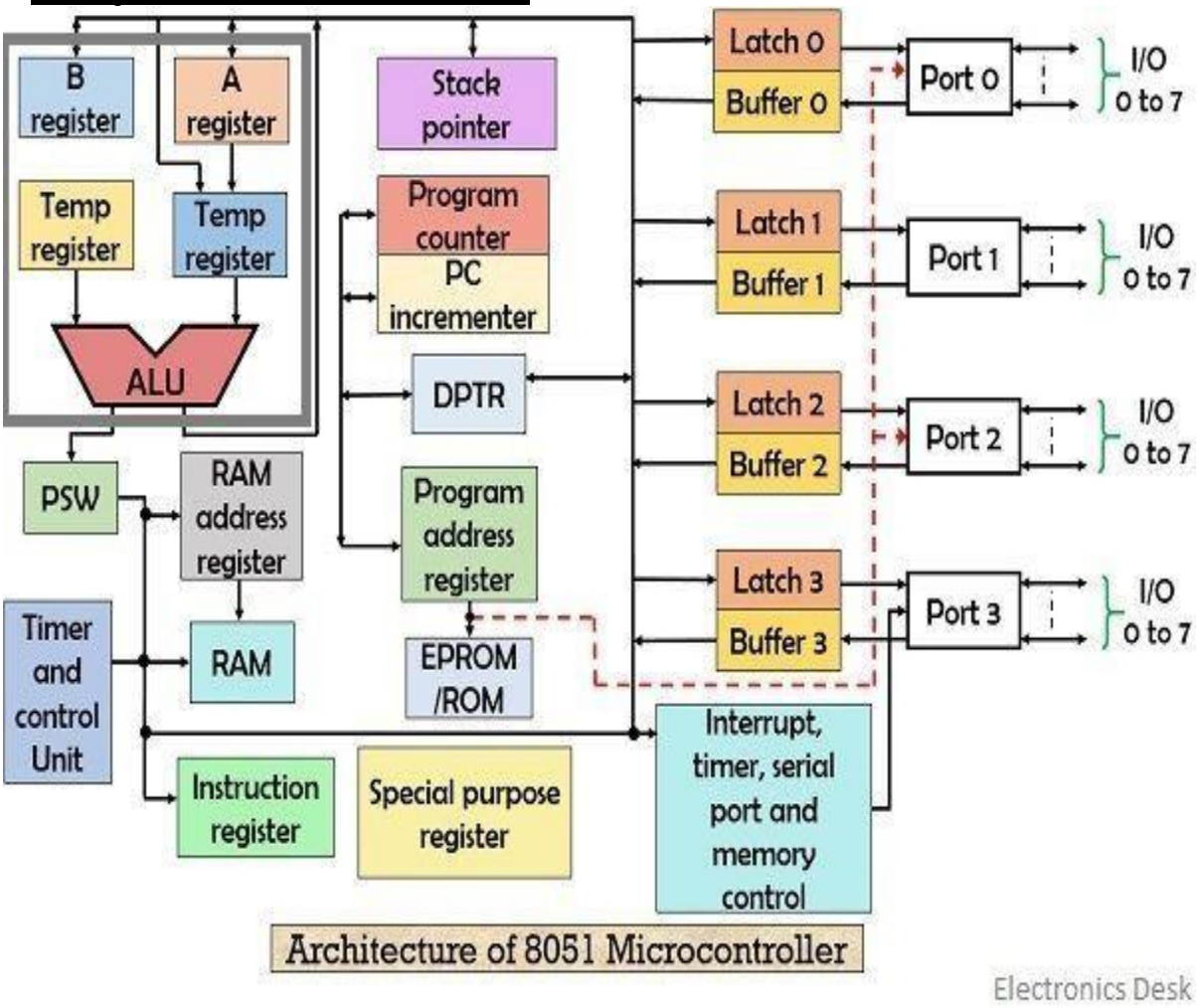


Figure 3.1 Internal Block diagram of mc 8051

8051 Microcontroller Hardware:

The 8051 microcontroller actually includes a whole family of microcontrollers that have numbers ranging from 8031 to 8751 and are available in N-Channel Metal Oxide Silicon (NMOS) and Complementary Metal Oxide Silicon (CMOS) construction in a variety of housed in a 40-pin DIP, and direct the investigation of a particular type to thedata books.

The block diagram of the 8051 in Figure 2. la shows all of the features unique to microcontrollers:

1. Internal ROM and RAM
2. I/O ports with programmable pins
3. Timers and counters
4. Serial data communication

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

The 8051 architecture consists of these specific features:

Eight-bit CPU with registers A (the accumulator) and B

Sixteen-bit program counter (PC) and data pointer (DPTR)

Eight-bit program status word (PSW)

Eight-bit stack pointer (SP)

Internal ROM or EPROM (8751) of 0 (8031) to 4K (8051)

Internal RAM of 128 bytes:

Four register banks, each containing eight registers

Sixteen bytes, which may be addressed at the bit level

Eighty bytes of general-purpose data memory

Thirty-two input/output pins arranged as four 8-bit ports: PO-P3

Two 16-bit timer/counters: TO and Tl

Full duplex serial data receiver/transmitter: SBUF

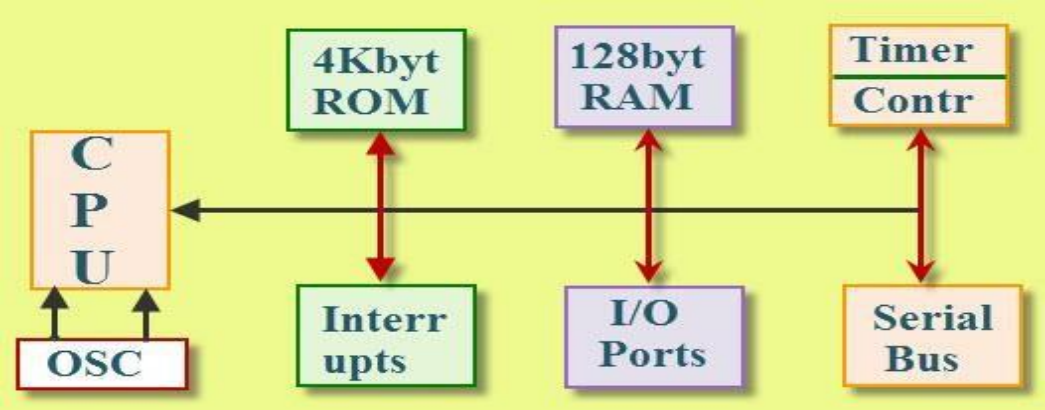


Figure 3.2 simplified block diagram of 8051

Control registers: TCON, TMOD, SCON, PCON, IP, and IE

Two external and three internal interrupt sources

Oscillator and clock circuits

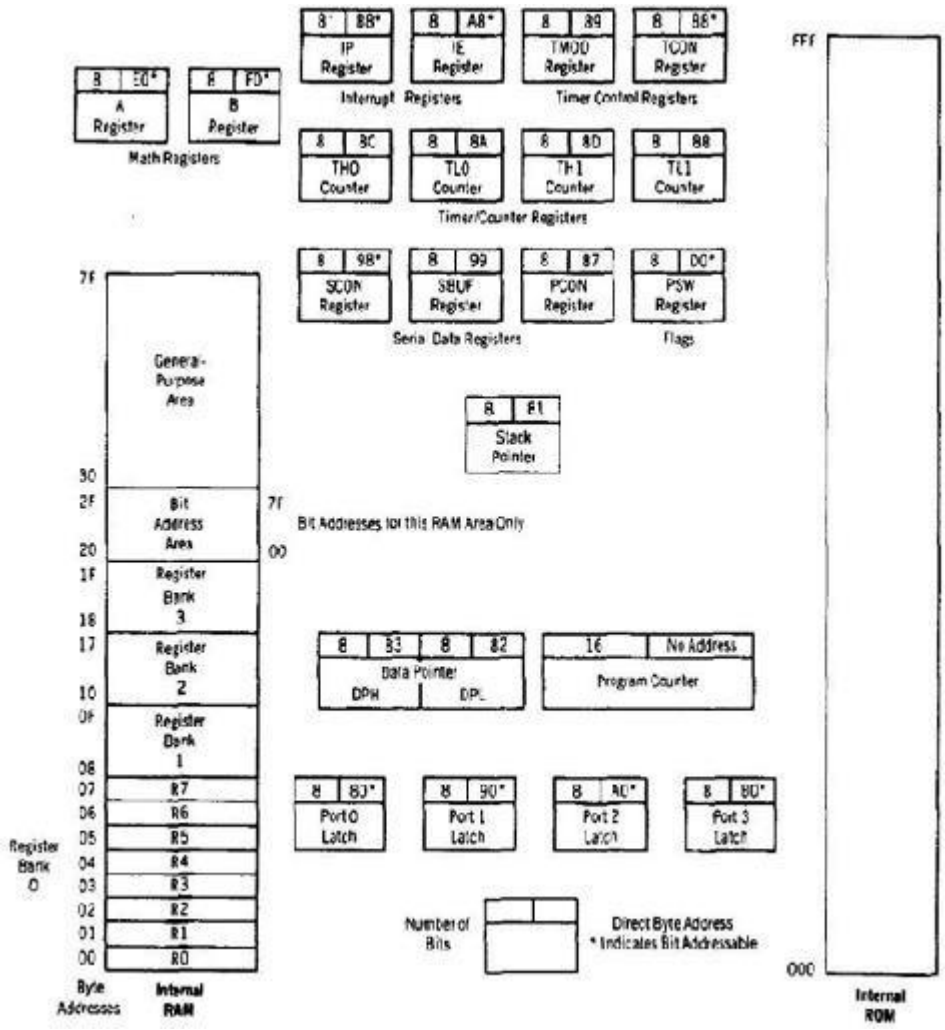


Figure 3.3 Programming model of 8051 Micro Controller

The programming model of the 8051 in Figure 2. Ib shows the 8051 as a collection of 8- and 16-bit registers and 8-bit memory locations. These registers and memory locations can be made to operate using the software instructions that are incorporated as part of the design. The program instructions have to do with the control of the registers and digital data paths that are physically contained inside the 8051, as well as memory locations that are physically located outside the 8051.

The model is complicated by the number of special-purpose registers that must be present to make a microcomputer a microcontroller. A cursory inspection of the

model is recommended for the first-time viewer; return to the model as needed while progressing through the remainder of the text.

Most of the registers have a specific function; those that do occupy an individual block with a symbolic name, such as A or THO or PC. Others, which are generally indistinguishable from each other, are grouped in a larger block, such as internal ROM or RAM memory.

Each register, with the exception of the program counter, has an internal 1-byte address assigned to it. Some registers (marked with an asterisk * in Figure 2.1b) are both byte and bit addressable. That is, the entire byte of data at such register addresses may be read or altered, or individual bits may be read or altered. Software instructions are generally able to specify a register by its address, its symbolic name, or both. A pinout of the 8051 packaged in a 40-pin DIP is shown in Figure 2.2 with the full and abbreviated names of the signals for each pin. It is important to note that many of the pins are used for more than one function (the alternate functions are shown in parentheses in Figure 2.2). Not all of the possible 8051 features may be used *at the same time.*

Programming instructions or physical pin connections determine the use of any multifunction pins. For example, port 3 bit 0 (abbreviated P3.0) may be used as a general purpose I/O pin, or as an input (RXD) to SBUF, the serial data receiver register. The system designer decides which of these two functions is to be used and designs the hardware and software affecting that pin accordingly.

Program Counter and Data Pointer

The 8051 contains two 16-bit registers: the program counter (PC) and the data pointer (DPTR). Each is used to hold the address of a byte in memory.

Program instruction bytes are fetched from locations in memory that are addressed by the PC. Program ROM may be on the chip at addresses OOOOh to OFFFh, external to the chip for addresses that exceed OFFFh, or totally external for all addresses from OOOOh to FFFFh. The PC is automatically incremented after every instruction byte is fetched and may also be altered by certain instructions. The PC is the only register that does not have an internal address.

The DPTR register is made up of two 8-bit registers, named DPH and DPL, that are

used to furnish memory addresses for internal and external code access and external data access. The DPTR is under the control of program instructions and can be specified by its 16-bit name, DPTR, or by each individual byte name, DPH and DPL. DPTR does not have a single internal address; DPH and DPL are each assigned an address.

A and B CPU Registers

The 8051 contains 34 general-purpose, or working, registers. Two of these, registers A and B, comprise the mathematical core of the 8051 central processing unit (CPU). The other 32 are arranged as part of internal RAM in four banks, BO-B3, of eight registers each, named RO to R7.

The A (accumulator) register is the most versatile of the two CPU registers and is used for many operations, including addition, subtraction, integer multiplication and division, and Boolean bit manipulations. The A register is also used for all data transfers between the 8051 and any external memory. The B register is used with the A register for multiplication and division operations and has no other function other than as a location where data may be stored.

Flags and the Program Status Word (PSW):

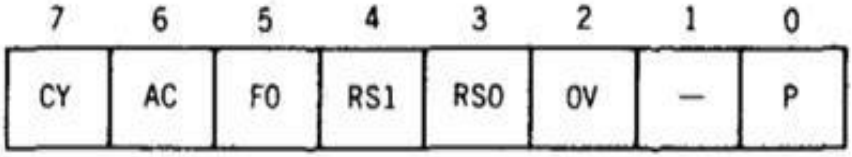
Flags are I -bit registers provided to store the results of certain program instructions. Other instructions can test the condition of the flags and make decisions based upon the flag states. In order that the flags may be conveniently addressed, they are grouped inside the program status word (PSW) and the power control (PCON) registers.

The 8051 has four math flags that respond automatically to the outcomes of math operations and three general-purpose user flags that can be set to I or cleared to 0 by the programmer as desired. The math flags include carry (C), auxiliary carry (AC), overflow (OV), and parity (P). User flags are named FO, GFO, and GF1;

they are general-purpose flags that may be used by the programmer to record some event in the program. Note that all of the flags can be set and cleared by the programmer at will. The math flags, however, are also affected by math operations.

The program status word is shown in Figure. The PSW contains the math flags, user program flag FO, and the register select bits that identify which of the four generalpurpose register banks is currently in use by the program. The remaining two user flags, GFO and GFl, are stored in PCON,

PSW Program Status Word Register



THE PROGRAM STATUS WORD (PSW) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	CY	Carry flag; used in arithmetic, JUMP, ROTATE, and BOOLEAN instructions
6	AC	Auxilliary carry flag; used for BCD arithmetic
5	F0	User flag 0
4	RS1	Register bank select bit 1
3	RS0	Register bank select bit 0

RS1	RS0	
0	0	Select register bank 0
0	1	Select register bank 1
1	0	Select register bank 2
1	1	Select register bank 3

2	OV	Overflow flag; used in arithmetic instructions
1	—	Reserved for future use
0	P	Parity flag; shows parity of register A: 1 = Odd Parity

Bit addressable as PSW.0 to PSW.7

Detailed descriptions of the math flag operations will be discussed in chapters that

cover the opcodes that affect the flags. The user flags can be set or cleared using data move instructions.

2. 3 Instruction sets

8051 has about 111 instructions. These can be grouped into the following categories

Arithmetic Instructions
Logical Instructions
Data Transfer instructions
Boolean Variable Instructions
Program Branching Instructions

The following nomenclatures for register, data, address and variables are used while writeinstructions

A: Accumulator

B: "B" register

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 all

conditional jumps.

bit: Directly addressed bit in internal RAM or SFR

Some Simple Instructions:

MOV dest, source; dest = source

MOV A, #72H;A=72H

MOV R4, #62H ; R4=62H
MO B,0F9H; B=the content of F9'th byte of RAM
MOV DPTR, #7634H

MOV DPL,#34H

MOV DPH, #76H

MOV P1, A ; mov A to port 1

Note 1:

MOV A , #72H
After instruction
"MOV A,72H " the content of 72'th byte of RAM replace in accumulator

MOV A, R3 ≡ MOV A, R3

```
ADD   A, Source          ;A=A+SOURCE
      ADD   A,#6         ;A=A+6
      ADD   A,R6         ;A=A+R6

      ADD   A,6          ;A=A+[6] or
                         A=A+R6
     ADD          A, 0F3H  ;  A=A+[0F3H]
SUBB  A, Source          ;A=A-SOURCE-C
      SUBB  A,#6         ;A=A-6
      SUBB  A,R6         ;A=A+R6
```

MUl & Div:

MUL AB ; B|A = A*B

MOV A, #25

MOV B, #65H

MUL AB

- DIV AB ;A = A/B,
- B = A mod B
- ;B=0EH, A=99H

```
MOV   A,#25
      MOV   B,#10
      DIV   AB          ;A=2, B=5
SETB  bit               ; bit=1
CLR   bit               ; bit=0
SETB  C                 ; CY=1
SETB  P0.0              ;bit 0 from port 0 =1
SETB  P3.7              ;bit 7 from port 3 =1
SETB  ACC.2             ;bit 2 from ACCUMULATOR
                        =1
SETB  05                ;set high D5 of RAM loc. 20h
```

Note:

CLR instruction is as same as SETB i.e.:

```
      CLR   C     ;CY=0
```

But following instruction is only for CLR:

```
      CLR   A     ;A=0
DEC   byte        ;byte=byte-1
INC   byte        ;byte=byte+1
INC   R7
DEC   A
DEC   40H         ; [40]=[40]-1
```

RR – RL – RRC – RLC A

EXAMPLE:

```
RR      A
```

RR:

RRC:

RL:

RLC:

ANL - ORL – XRL

Bitwise

Logical

Operations:

AND, OR,

XOR

EXAMPLE:

```
MOV   R5,#89H
      ANL   R5,#08H
```

CPL A ;1's complement

Example:

```
      MOV A,#55H ;A=01010101 B
L01:  CPL A
      MOV     P1,A
      ACALL   DELAY
      SJMP    L01
```

3.4 addressing modes

Addressing modes of 8051

In 8051 There are six types of addressing modes.

- Immediate Addressing Mode
- Register Addressing Mode

- Direct Addressing Mode
- Register Indirect Addressing Mode
- Indexed Addressing Mode
- Implied Addressing Mode

Immediate addressing mode

In this Immediate Addressing Mode, the data is provided in the instruction itself. The data is provided immediately after the opcode. These are some examples of Immediate Addressing Mode

MOV A, #0AFH

MOV R3, #45H

MOV DPTR, #FE00H

In these instructions, the # symbol is used for immediate data. In the last instruction, there is DPTR. The DPTR stands for Data Pointer. Using this, it points the external data memory location. In the first instruction, the immediate data is AFH, but one 0 is added at the beginning. So when the data is starting with A to F, the data should be preceded by 0.

Register addressing mode

In the register addressing mode the source or destination data should be present in a register (R0 to R7). These are some examples of Register Addressing Mode.

MOV A, R5

MOV R2, #45H

MOV RO, A

In 8051, there is no instruction like **MOVR5, R7**. But we can get the same result by using this instruction **MOV R5, 07H**, or by using **MOV 05H, R7**. But this two instruction will work when the selected register bank is **RB0**. To use another register bank and to get the same effect, we have to add the starting address of that register bank with the register number. For an example, if the RB2 is selected, and we want to access R5, then the address will be (10H + 05H = 15H), so the instruction will look like this **MOV 15H, R7**. Here 10H is the starting address of Register Bank 2.

Direct Addressing Mode

In the Direct Addressing Mode, the source or destination address is specified by using 8-bit data in the instruction. Only the internal data memory can be used in this mode. Here some of the examples of direct Addressing Mode.

MOV 80H, R2

MOV R6, 60H

MOV R0, 05H

The first instruction will send the content of registerR6 to port P0 (Address of Port 0 is 80H). The second one is forgetting content from 45H to R2. The third one is used to get data from Register R5 (When register bank RB0 is selected) to register R5.

Register indirect addressing Mode

In this mode, the source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes. Let us see some examples of this mode.

MOV 0E5, @R0

MOV @R6, P0

MOV @R2, 60H

In the instructions, the @ symbol is used for register indirect addressing. In the first instruction, it is showing that theR0 register is used. If the content of R0 is 40H, then that instruction will take the data which is located at location 40H of the internal RAM. In the second one, if the content of R6 is 30H, then it indicates that the content of port P0 will be stored at location 30H in the internal RAM

MOV A, @RI

MOVX @DPTR,, A

In these two instructions, the X in MOVX indicates the external data memory. The external data memory can only be accessed in register indirect mode. In the first instruction if the R0 is holding 40H, then A will get the content of external RAM location40H. And in the second one, the content of A is overwritten in the location pointed by DPTR.

Indexed addressing mode

In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A. These are some examples of Indexed addressing mode.

MOVC A, @A+PC

MOVC A, @A+ DPTR

The C in MOVC instruction refers to code byte. For the first instruction, let us consider A holds 30H. And the PC value is1125H. The contents of program memory location 1155H (30H + 1125H) are moved to register A

Implied Addressing Mode

In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only. These types of instructions are also known as register specific instruction. Here are some examples of Implied Addressing Mode.

RLA

SWAP A

These are 1- byte instruction. The first one is used to rotate the A register content to the Left. The second one is used to swap the nibbles in A.

3.5 interrupt structure

Interrupts:

1. Enabling and Disabling Interrupts
2. Interrupt Priority
3. Writing the ISR (Interrupt Service Routine)

Interrupt Enable (IE) Register :

- EA : Global enable/disable.
- --- : Undefined.
- ET2 :Enable Timer 2 interrupt.
- ES :Enable Serial port interrupt.
- ET1 :Enable Timer 1 interrupt.
- EX1 :Enable External 1 interrupt.
- ET0 : Enable Timer 0 interrupt.

- EX0 : Enable External 0 interrupt.

EA	—	ET2	ES	ET1	EX1	ET0	EX0

Interrupt Vectors:

Interrupt	Vector Address
System Reset	0000H
External 0	0003H
Timer 0	000BH
External 1	0013H
Timer 1	001BH
Serial Port	0023H
Timer 2	002BH

3.6 Timer Peripheral Control RegistersPCON (Power Control)

The PCON or Power Control register, as the name suggests is used to control the 8051 Microcontroller's Power Modes and is located at 87H of the SFR Memory Space. Using two bitsin the PCON Register, the microcontroller can be set to Idle Mode and Power Down Mode.

During Idle Mode, the Microcontroller will stop the Clock Signal to the ALU (CPU) but it is given to other peripherals like Timer, Serial, Interrupts, etc. In order to terminate the Idle Mode,you have to use an Interrupt or Hardware Reset.

In the Power Down Mode, the oscillator will be stopped and the power will be reduced to 2V. Toterminate the Power Down Mode, you have to use the Hardware Reset.

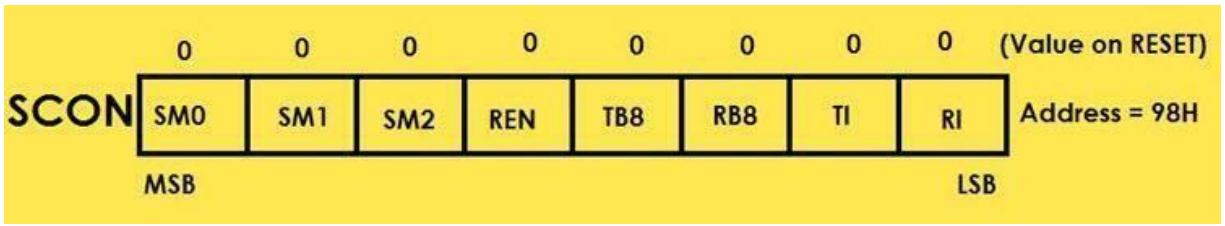
Apart from these two, the PCON Register can also be used for few additional purposes. TheSMOD Bit in the PCON Register is used to control the Baud Rate of the Serial Port.

There are two general purpose Flag Bits in the PCON Register, which can be used by theprogrammer during execution.

SCON (Serial Control)

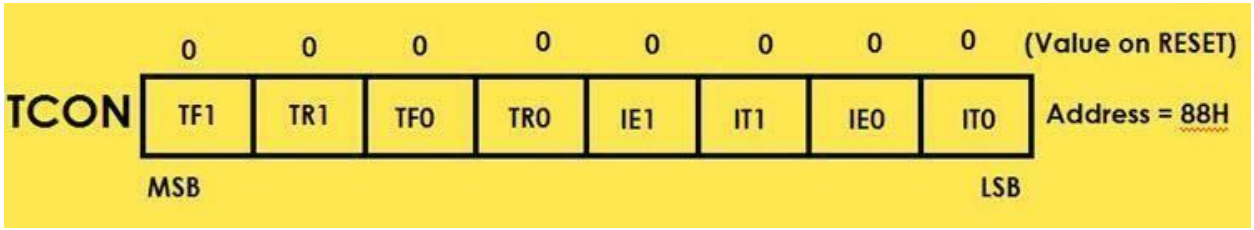
The Serial Control or SCON SFR is used to control the 8051 Microcontroller's Serial Port. It islocated as an address of 98H. Using SCON, you can control the Operation Modes of the Serial Port, Baud Rate of the Serial Port and Send or Receive Data using Serial Port.

SCON Register also consists of bits that are automatically SET when a byte of data is transmittedor received.



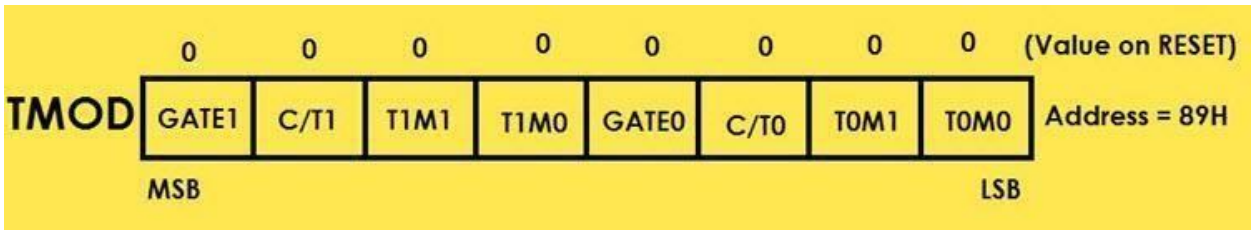
TCON (Timer Control)

Timer Control or TCON Register is used to start or stop the Timers of 8051 Microcontroller. Italso contains bits to indicate if the Timers has overflowed. The TCON SFR also consists of Interrupt related bits.



TMOD (Timer Mode)

The TMOD or Timer Mode register or SFR is used to set the Operating Modes of the Timers T0and T1. The lower four bits are used to configure Timer0 and the higher four bits are used to configure Timer1.



The Gatex bit is used to operate the Timerx with respect to the INTx pin or regardless of theINTx pin.

GATE1 = 1 ==> Timer1 is operated only if

INT1 is SET. GATE1 = 0 ==> Timer1 is

operates irrespective of INT1 pin.GATE0 = 1

==> Timer0 is operated only if INT0 is SET.

GATE0 = 0 ==> Timer0 is operates

irrespective of INT0 pin.

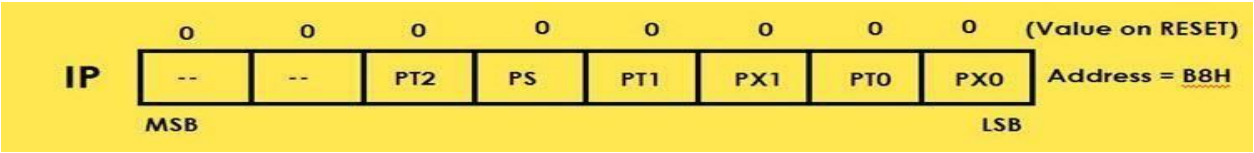
The C/Tx bit is used selects the source of pulses for the Timer to count. C/T1 = 1 ==> Timer1 counts pulses from Pin T1 (P3.5) (Counter Mode) C/T1 = 0 ==> Timer1 counts pulses from internal oscillator Timer Mode)

C/T0 = 1 ==> Timer0 counts pulses from Pin T0 (P3.4) (Counter Mode) C/T0 = 0 ==> Timer0 counts pulses from internal oscillator (Timer Mode)

TxM0	TxM1	Mode	Description
0	0	0	13-bit Timer Mode (THx – 8-bit and TLx – 5-bit)
0	1	1	16-bit Timer Mode
1	0	2	8-bit Auto Reload Timer Mode
1	1	3	Two 8-bit Timer Mode or Split Timer Mode

IP (Interrupt Priority)

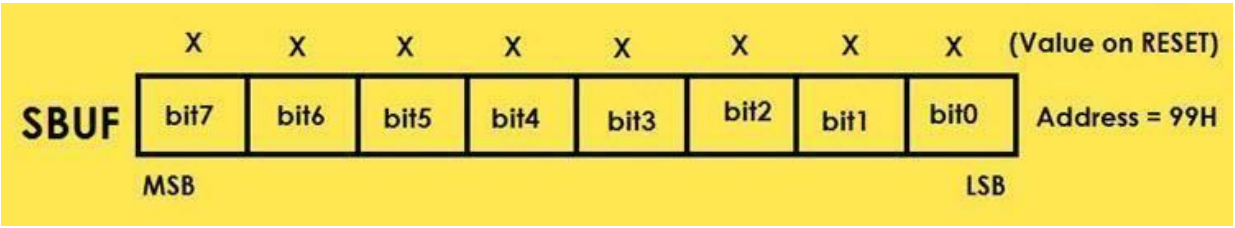
The IP or Interrupt Priority Register is used to set the priority of the interrupt as High or Low. If a bit is CLEARED, the corresponding interrupt is assigned low priority and if the bit is SET, theinterrupt is assigned high priority.



Peripheral Data Registers

SBUF (Serial Data Buffer)

The Serial Buffer or SBUF register is used to hold the serial data while transmission orreception.



3.7 I/O Ports

Internal Memory:

A functioning computer must have memory for program code bytes, commonly in ROM, and RAM memory for variable data that can be altered as the program runs. The 8051 has internal RAM and ROM memory for these functions. Additional memory can be added externally using suitable circuits.

Unlike microcontrollers with Von Neumann architectures, which can use a *single* memory address for either program code or data, *but not for both*, the 8051 has a

Harvard architecture, which uses the *same address,* in *different* memories, for code and data. Internal circuitry accesses the correct memory based upon the nature of the operation in progress.

Internal RAM:

The 128-byte internal RAM, which is shown generally in Figure 2.1 and in detail in Figure 2.5, is organized into three distinct areas:

2. Thirty-two bytes from address OOh to I Fh that make up 32 working registers organized as four banks of eight registers each. The four register banks are numbered 0 to 3 and are made up of eight registers named RO to R7. Each register
can be addressed by name (when its bank is selected) or by its RAM address. Thus RO of bank 3 is RO (if bank 3 is currently selected) or address 18h (whether
bank 3 is selected or not). Bits RSO and RSI in the PSW determine which bank of registers is currently in use at any time when the program is running. Register banks not selected can be used as general-purpose RAM. Bank 0 is selected upon reset.

3. A Wf-addressable area of 16 bytes occupies RAM *byte* addresses 20h to 2Fh, forming a total of 128 addressable bits. An addressable bit may be specified by its *bit* address of OOh to 7Fh, or 8 bits may form any *byte* address from 20h to 2Fh. Thus, for example, bit address 4Fh is also bit 7 of byte address 29h. Addressable bits are useful when the program need only remember a binary event (switch on, light off, etc.). Internal RAM is in short supply as it is, so why use a byte when a bit will do?

4. A general-purpose RAM area above the bit area, from 30h to 7Fh, addressable as bytes.

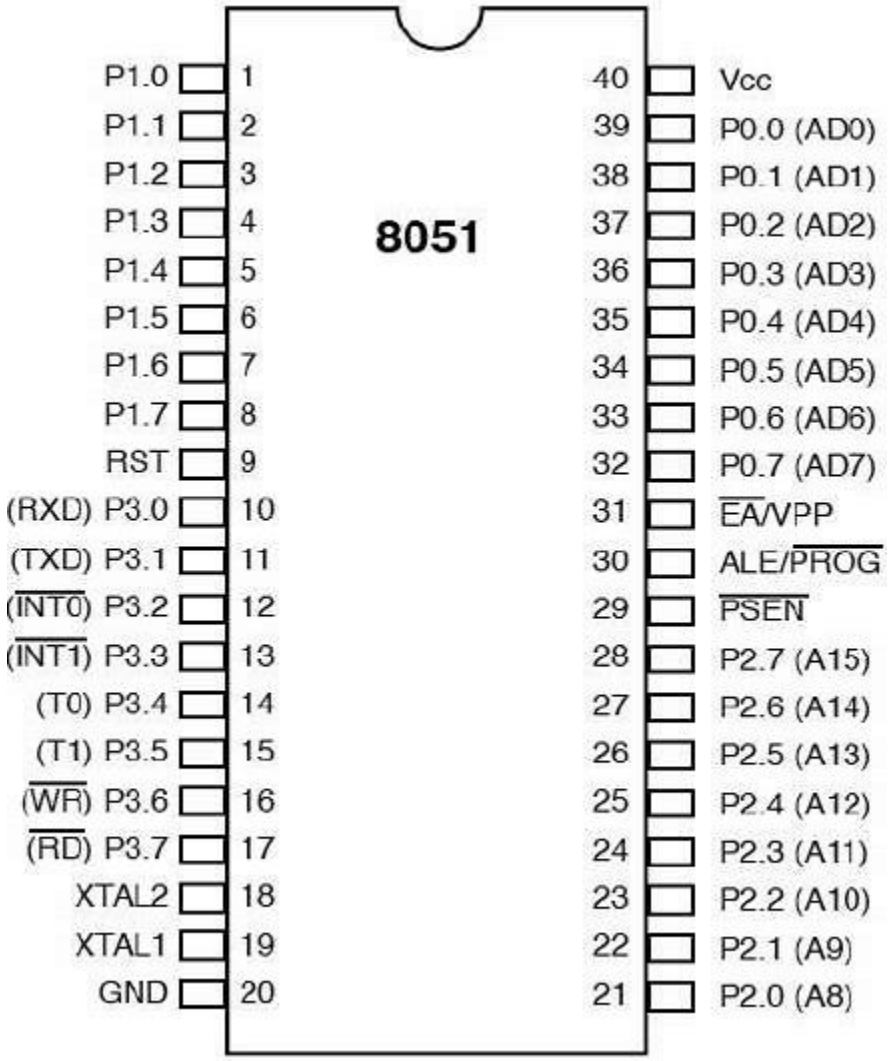


Figure Pin diagram of 8051

PortO:

Port 0 pins may serve as inputs, outputs, or, when used together, as a bi-directional loworder address and data bus for external memory. For example, when a pin is to be used as an input, a 1 *must be* written to the corresponding port 0 latch by the program, thus turning both of the output transistors off, which in turn causes the pin to "float" in a highimpedance state, and the pin is essentially

connected to the input buffer.

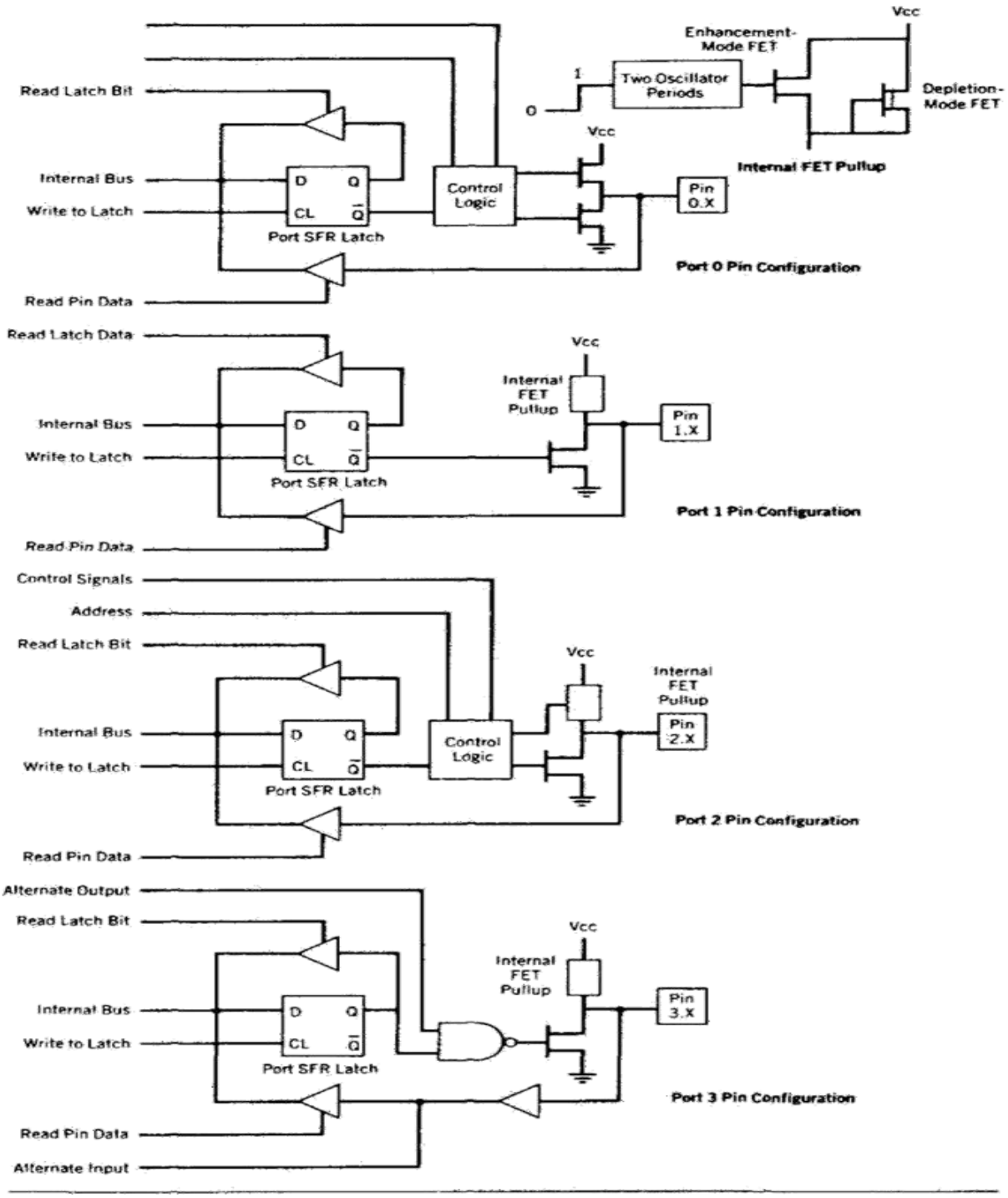
When used as an output, the pin latches that are programmed to a 0 will turn on the

lower FET, grounding the pin. All latches that are programmed to a 1 still float; thus, external pull up resistors will be needed to supply a logic high when using port 0 as an output.

When port 0 is used as an address bus to external memory, internal control signals

switch the address lines to the gates of the Field Effect Transistories (FETs). A logic I on an address bit will turn the upper FET on and the lower FET off to provide a logic high at the pin. When the address bit is a zero, the lower FET is on and the upper FET off to provide a logic low at the pin.

Port Pin Circuits



After the address has been formed and latched into external circuits by the Address Latch Enable (ALE) pulse, the bus is turned around to become a data bus. Port 0 now reads data from the external memory and must be configured as an input, so a logic 1 is automatically written by internal control logic to all port 0 latches.

Port l

Port 1 pins have no dual functions. Therefore, the output latch is connected directly to the gate of the lower FET, which has an FET circuit labeled "Internal FET Pull up" as an active pull up load.

Used as an input, a 1 is written to the latch, turning the lower FET off; the pinand the input to the pin buffer are pulled high by the FET load. An external circuit can overcome the high impedance pull up and drive the pin low to input a 0 or leave the input high for a 1.

If used as an output, the latches containing a I can drive the input of an external circuit high through the pull up. If a 0 is written to the latch, the lower FET is on, the pull up is off, and the pin can drive the input of the external circuit low.

To aid in speeding up switching times when the pin is used as an output, the internal FET pull up has another FET in parallel with it. The second FET is turned on for two oscillator time periods during a low-to-high transition on the pin, as shown in Figure 2.7.

This arrangement provides a low impedance path to the positive voltage supply to help reduce rise times in charging any parasitic capacitances in the external circuitry.

Port 2

Port 2 may be used as an input/output port similar in operation to port 1. The alternate use of port 2 is to supply a high-order address byte in conjunction with the port 0 low-order byte to address external memory.

Port 2 pins are momentarily changed by the address control signals when supplying the high byte of a 16-bit address. Port 2 latches remain stable when

external memory is addressed, as they do not have to be turned around (set to 1) for data input as is the case for port 0.

Port3

Port 3 is an input/output port similar to port I. The input and output functions can

be programmed under the control of the P3 latches or under the control of various other special function registers. The port 3 alternate uses are shown in the following table:-

PIN	ALTERNATE USE	SFR
P3.0–RXD	Serial data input	SBUF
P3.1–TXD	Serial data output	SBUF
P3.2–$\overline{\text{INT0}}$	External interrupt 0	TCON.1
P3.3–$\overline{\text{INT1}}$	External interrupt 1	TCON.3
P3.4–T0	External timer 0 input	TMOD
P3.5–T1	External timer 1 input	TMOD
P3.6–$\overline{\text{WR}}$	External memory write pulse	—
P3.7–$\overline{\text{RD}}$	External memory read pulse	—

Unlike ports 0 and 2, which can have external addressing functions and change all

eight port bits when in alternate use, each pin of port 3 may be individually programmed to be used either as I/O or as one of the alternate functions.

External Memory

The system designer is not limited by the amount of internal RAM and ROM available on chip. Two separate external memory spaces are made available by the 16-bit PC and DPTR and by different control pins for enabling external ROM

and RAM chips. Internal control circuitry accesses the correct physical memory, depending upon the machine cycle state and the op code being executed.

There are several reasons for adding external memory, particularly program memory, when applying the 8051 in a system. When the project is in the prototype stage, the expense—in time and money—of having a masked internal ROM made for each program "try" is prohibitive.

To alleviate this problem, the manufacturers make available an EPROM version, the 8751, which has 4K of on-chip EPROM that may be programmed and erased as needed as the program is developed. The resulting circuit board layout will be identical to one that uses a factory-programmed 8051. The only drawbacks to the 8751 are the specialized EPROM programmers that must be used to program the non-standard 40-pin part, and the limit of "only" 4096 bytes of program code. The 8751 solution works well if the program will fit into 4K bytes. Unfortunately, many times, particularly if the program is written in a high-level language, the program size exceeds 4K bytes, and an external program memory is needed. Again, the manufacturers provide a version for the job, the ROMIess 8031. The EA pin is grounded when using the 8031, and all program code is contained in an external EPROM that may be as large as 64K bytes and that can be programmed using standard EPROM programmers.

External RAM, which is accessed by the DPTR, may also be needed when 128 bytes of internal data storage is not sufficient. External RAM, up to 64K bytes, may also be added to any chip in the 8051 family.

Connecting External Memory

Figure 2.8 shows the connections between an 8031 and an external memory configuration consisting of I6K bytes of EPROM and 8K bytes of static RAM. The 8051 accesses external RAM whenever certain program instructions are executed. External ROM is accessed whenever the EA (external access) pin is connected to ground or when the PC contains an address higher than the last address in the internal 4K bytes ROM (OFFFh). 8051 designs can thus use

internal and external ROM automatically; the 8031, having no internal ROM, must have EA grounded.

Figure 2.9 shows the timing associated with an external memory access cycle. During any memory access cycle, port 0 is time multiplexed. That is, it first provides the lower byte of the 16-bit memory address, then acts as a bidirectional

data bus to write or read a byte of memory data. Port 2 provides the high byte of the memory address during the entire memory read/write cycle.

The lower address byte from port 0 must be latched into an external register to save

the byte. Address byte save is accomplished by the ALE clock pulse that provides the correct timing for the '373 type data latch. The port 0 pins then become free to serve as a data bus.

If the memory access is for a byte of program code in the ROM, the PSEN (program store enable) pin will go low to enable the ROM to place a byte of program code on the data bus. If the access is for a RAM byte, the WR (write) or RD (read) pins will go low, enabling data to flow between the RAM and the data bus.

The ROM may be expanded to 64K by using a 27512 type EPROM and connecting the remaining port 2 upper address lines AI4-A15 to the chip.

At this time the largest static RAMs available are 32K in size; RAM can be expanded to 64K by using two 32K RAMs that are connected through address A14 of port 2. The

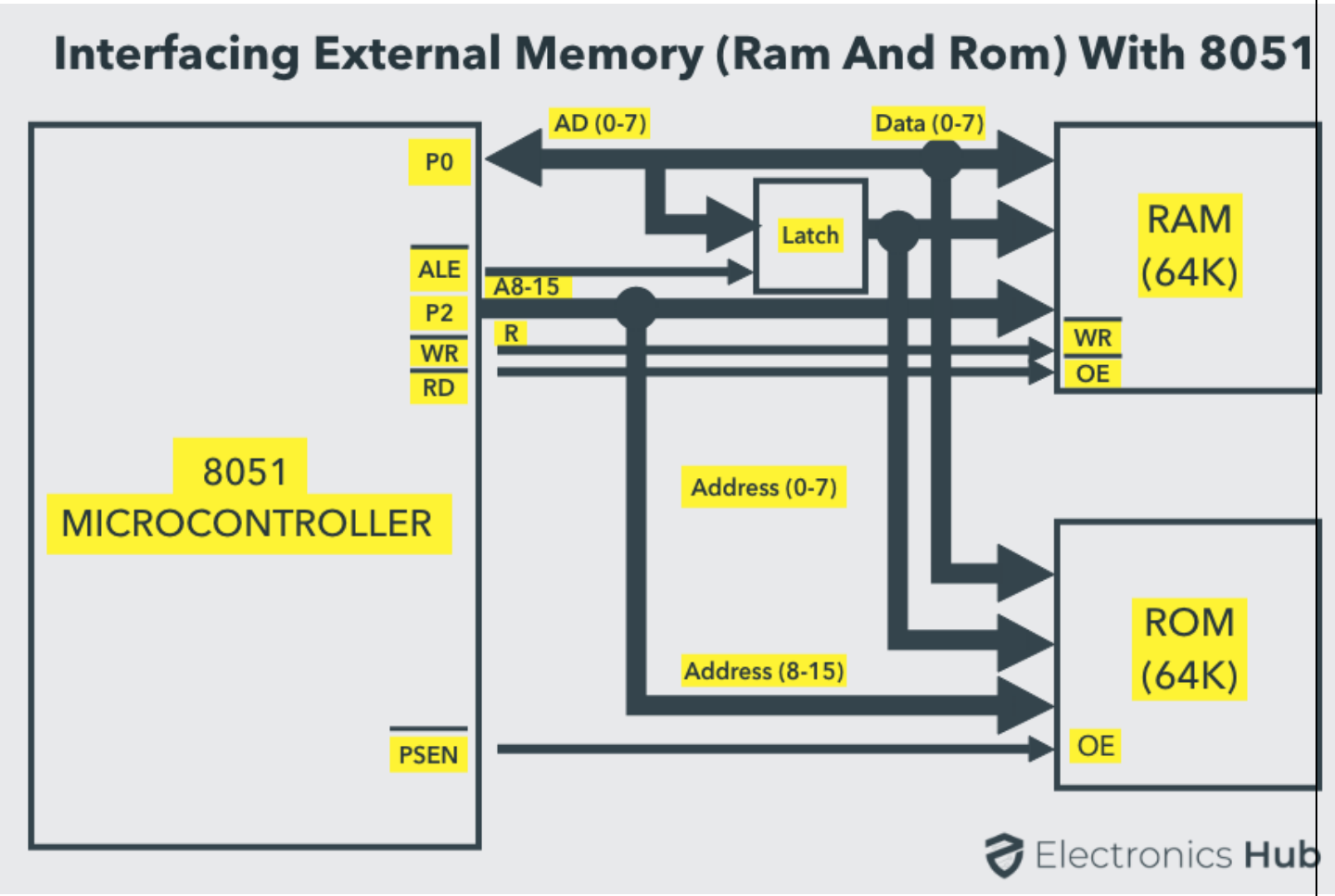


Figure Interfacing 8086 with external memory

first 32K RAM (OOOOh-7FFFh) can then be enabled when AI5 of port 2 is low, and the second 32K RAM (SOOOh-FFFFh) when A15 is high, by using an inverter.

Note that the WR and RD signals are alternate uses for port 3 pins 16 and 17. Also,

port 0 is used for the lower address byte and data; port 2 is used for upper address bits. The use of external memory consumes many of the port pins, leaving only port 1 and parts of port 3 for general I/O.

3.8 serial communication.

The 8051 microcontroller is parallel device that transfers eight bits of data simultaneously over eight data lines to parallel I/O devices. Parallel data transfer over a long is very expensive. Hence, a serial communication is widely used in long distance communication. In serial data communication, 8-bit data is converted to serial bits using a parallel in serial out shift register and then it is transmitted over a single data line. The data byte is always transmitted with least significant bit first.

BASICS OF SERIAL DATA COMMUNICATION, Communication Links

1. Simplex communication link: In simplex transmission, the line is dedicated for transmission. The transmitter sends and the receiver receives the data.

2. Half duplex communication link: In half duplex, the communication link can be used for either transmission or reception. Data is transmitted in only one direction at a time. Receiver Transmitter

3. Full duplex communication link: If the data is transmitted in both ways at the same time, it is a full duplex i.e. transmission and reception can proceed simultaneously. This communication link requires two wires for data, one for transmission and one for reception.

Types of Serial communication: Serial data communication uses two types of communication.

1. Synchronous serial data communication: In this transmitter and receiver are synchronized. It uses a common clock to synchronize the receiver and the transmitter. First the synch character is sent and then the data is transmitted. This format is generally used for high speed transmission. In Synchronous serial data communication a block of data is transmitted at a time. Receiver Transmitter Transmitter Start D0 D1 D2 D3 D4 D5 D6 D7 D8 Stop Receiver

2. Asynchronous Serial data transmission: In this, different clock sources are used for transmitter and receiver. In this mode, data is transmitted with start and stop bits. A transmission begins with start bit, followed by data and then stop bit. For error checking purpose parity bit is included just prior to stop bit. In

Asynchronous serial data communication a single byte is transmitted at a time. Data Clock 1 Clock2 Baud rate:

The rate at which the data is transmitted is called baud or transfer rate. The baud rate is the reciprocal of the time to send one bit. In asynchronous transmission, baud rate is not equal to number of bits per second. This is because; each byte is preceded by a start bit and followed by parity and stop bit. For example, in synchronous transmission, if data is transmitted with 9600 baud, it means that 9600 bits are transmitted in one second. For bit transmission time = 1 second/ 9600 = 0.104 ms. Over the years, dozens of serial protocols have been crafted to meet particular needs of embedded systems.

1. USB (universal serial bus), and Ethernet, are a couple of the well- known computing serial interfaces.

2. Other very common serial interfaces include SPI, I2C, RS-232 and so on.

3. Each of these serial interfaces can be sorted into one of two groups: synchronous or asynchronous.

4. A Synchronous serial interface always pairs its data line(s) with a clock signal, so all devices on a synchronous serial bus share a common clock. This often allows faster serial transfer, but it also requires at least one extra wire between communicating devices. Examples of synchronous interfaces include SPI, and I2C. Asynchronous means that data is transferred without support from an external clock signal. Minimizes the required wires and I/O pins, but we need to put some extra effort into reliably transferring and receiving data. 5. The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications subsystem. UART is also a common integrated feature in most microcontrollers. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Communication can be “full duplex” (both send and receive at the same time) or “half duplex” (devices take turns transmitting and receiving). 6. The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) controller is another implementation of the serial port. UART supports only asynchronous mode, whereas USART supports both

asynchronous and synchronous modes. Unlike Ethernet, Firewire etc., there is no specific port for UART/USART. They are commonly used in conjugation with protocols like RS- 232, RS-434 etc.

SCON (Serial Control)

The Serial Control or SCON SFR is used to control the 8051 Microcontroller's Serial Port. It islocated as an address of 98H. Using SCON, you can control the Operation Modes of the Serial Port, Baud Rate of the Serial Port and Send or Receive Data using Serial Port.

SCON Register also consists of bits that are automatically SET when a byte of data is transmittedor received.

Rules for UART The asynchronous serial protocol has a number of built-in rules - mechanisms that help ensure robust and error-free data transfers. These mechanisms compensate the non-existence of the common clock signal: Baud Rate: The baud rate specifies how fast data is sent over a serial line. It's usually expressed in units of bits-per-second (bps).• This value determines how long the transmitter holds a serial line high/low or at• what period the receiving device samples it's line. Baud rates can be just about any value within reason. The only requirement is that• both devices operate at the same rate. Common baud rate is 9600 bps. Other "standard" baud are 1200, 2400, 4800,• 19200, 38400, 57600, and 115200. Framing the data Each block (usually a byte) of data transmitted is actually sent in a packet or frame of• bits. Frames are created by appending synchronization and parity bits to our data.• Some symbols in the frame have configurable bit sizes.• Data chunk :The amount of data in each packet can be set to anything from 5 to 9• bits. Certainly, the standard data size is your basic 8-bit byte, but other sizes have their uses. A 7-bit data chunk can be more efficient than 8, especially if you're just transferring 7-bit ASCII characters. After agreeing on a character-length, both serial devices also have to agree on the endianness of their data. Is data sent mostsignificant bit (msb) to least, or vice-versa? If it's not otherwise stated, you can usually assume that data is transferred least-significant bit (lsb) first. Synchronization bits: The synchronization bits are two or three special bits• transferred with each chunk of data. They are the start bit and the stop bit(s), and mark the beginning and end of a packet. There is always only one start bit, but the number of stop bits is configurable to either one or two (though it's commonly left at one). The start bit is always indicated by an idle data line going from 1 to 0, while the stop bit(s) will transition back to the idle state by holding the line at 1. Parity bits: Parity is optional, and not very widely used. It can be helpful for• transmitting across noisy mediums, but it'll also slow down data transfer a bit and requires both sender and receiver to implement error-handling (usually, received data that fails must be re-sent). A variety of communication protocols have been developed based on serial communication in the past few decades. Some of them are: SPI – Serial Peripheral Interface: It is a

three-wire based communication system. One wire each• for Master to slave and Vice-versa, and one for clock pulses. There is an additional SS (Slave Select) line, which is mostly used when we want to send/receive data between multiple ICs. I2C – Inter-Integrated Circuit : Pronounced eye-two-see or eye-square-see, this is an advanced• form of USART. The transmission speeds can be as high as 400KHz. The I2C bus has two wires – one for clock, and the other is the data line, which is bi-directional – this being the reason it is also sometimes (not always – there are a few conditions) called Two Wire Interface (TWI). It is a new and revolutionary technology invented by Philips. FireWire – Developed by Apple. High-speed buses capable of audio/video transmission. The bus• contains a number of wires depending upon the port, which can be either a 4-pin one, or a 6-pin one, or an 8-pin one. Ethernet : Used mostly in LAN connections, the bus consists of 8 lines, or 4 Tx/Rx pairs.• Universal serial bus (USB). Most popular of all serial interfaces. Is used for virtually all type of• connections. The bus has 4 lines: VCC, Ground, Data+, and Data-.

8051 SERIAL COMMUNICATION

The 8051 supports a full duplex serial port. Three special function registers support serial communication.

1. SBUF Register: Serial Buffer (SBUF) register is an 8-bit register. It has separate SBUF registers for data transmission and for data reception. For a byte of data to be transferred via the TXD line, it must be placed in SBUF register. Similarly, SBUF holds the 8-bit data received by the RXD pin and read to accept the received data.

2. SCON register: The contents of the Serial Control (SCON) register are shown below. This register contains mode selection bits, serial port interrupt bit (TI and RI) and also the ninth data bit for transmission and reception (TB8 and RB8).

3. PCON register: The SMOD bit (bit 7) of PCON register controls the baud rate in asynchronous mode transmission.

SERIAL COMMUNICATION MODES Mode 0 In this mode serial port runs in synchronous mode. The data is transmitted and received through RXD pin and TXD is used for clock output.

In this mode the baud rate is 1/12 of clock frequency. Mode 1 In this mode SBUF becomes a 10 bit full duplex transceiver. The ten bits are 1 start bit, 8 data bit and 1 stop bit. The interrupt flag TI/RI will be set once transmission or reception is over. In this mode the baud rate is variable and is determined by the timer 1 overflow rate. Baud rate = [2smod/32] x Timer 1 overflow Rate = [2smod/32] x [Oscillator Clock Frequency] / [12 x [256 – [TH1]]] Mode 2 This is similar to mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bit, a programmable 9th data bit, 1 stop bit. Baud rate = [2smod/64] x Oscillator Clock Frequency Mode 3 This is similar to mode 2 except baud rate is calculated as in mode 1

3.9 Data transfer, manipulation, Control and I/O instructions

Data Manipulation Instructions :

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer usually divided into three basic types as follows.

1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions

Let's discuss one by one.

1. **Arithmetic instructions :**
   The four basic arithmetic operations are addition, subtraction, multiplication, and division. Most computers provide instructions for all four operations.

   Typical Arithmetic Instructions –

Name	Mnemonic	Example	Explanation
Increment	INC	INC B	It will increment the register B by 1 B<-B+1
Decrement	DEC	DEC B	It will decrement the register B by 1 B<-B-1
Add	ADD	ADD B	It will add contents of register B to the contents of the accumulator and store the result in the accumulator

			AC<-AC+B
Subtract	SUB	SUB B	It will subtract the contents of register B from the contents of the accumulator and store the result in the accumulator AC<-AC-B
Multiply	MUL	MUL B	It will multiply the contents of register B with the contents of the accumulator and store the result in the accumulator AC<-AC*B
Divide	DIV	DIV B	It will divide the contents of register B with the contents of the accumulator and store the quotient in the accumulator AC<-AC/B
Add with carry	ADDC	ADDC B	It will add the contents of register B and the carry flag with the contents of the accumulator and store the result in the accumulator AC<-AC+B+Carry flag
Subtract with borrow	SUBB	SUBB B	It will subtract the contents of register B and the carry flag from the contents of the accumulator and store the result in the accumulator

			AC<-AC-B-Carry flag
Negate(2's complement)	NEG	NEG B	It will negate a value by finding 2's complement of its single operand. This means simply operand by -1. B<-B'+1

2. Logical and Bit Manipulation Instructions :

Logical instructions perform binary operations on strings of bits stored in registers. They are useful for manipulating individual bits or a group of bits.

Typical Logical and Bit Manipulation Instructions –

Name	Mnemonic	Example	Explanation
Clear	CLR	CLR	It will set the accumulator to 0 AC<-0
Complement	COM	COM A	It will complement the accumulator AC<-(AC)'
AND	AND	AND B	It will AND the contents of register B with the contents of accumulator and store it in the accumulator AC<-AC AND B
OR	OR	OR B	It will OR the contents of register B with the contents of accumulator and store it in the accumulator AC<-AC OR B
Exclusive-OR	XOR	XOR B	It will XOR the contents of register B with the contents of the accumulator and store it in the accumulator AC<-AC XOR B

Clear carry	CLRC	CLRC	It will set the carry flag to 0 Carry flag<-0
Set carry	SETC	SETC	It will set the carry flag to 1 Carry flag<-1
Complement carry	COMC	COMC	It will complement the carry flag Carry flag<- (Carry flag)'
Enable interrupt	EI	EI	It will enable the interrupt
Disable interrupt	DI	DI	It will disable the interrupt

3. Shift Instructions :

Shifts are operations in which the bits of a word are moved to the left or right. Shift instructions may specify either logical shifts, arithmetic shifts, or rotate-type operations.

Typical Shift Instructions –

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

3. 10 simple programming exercises key board and display interface

The key board here we are interfacing 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 board. So only two ports of 8051 can be easily connected to the rows and columns of the key board.

When ever 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.

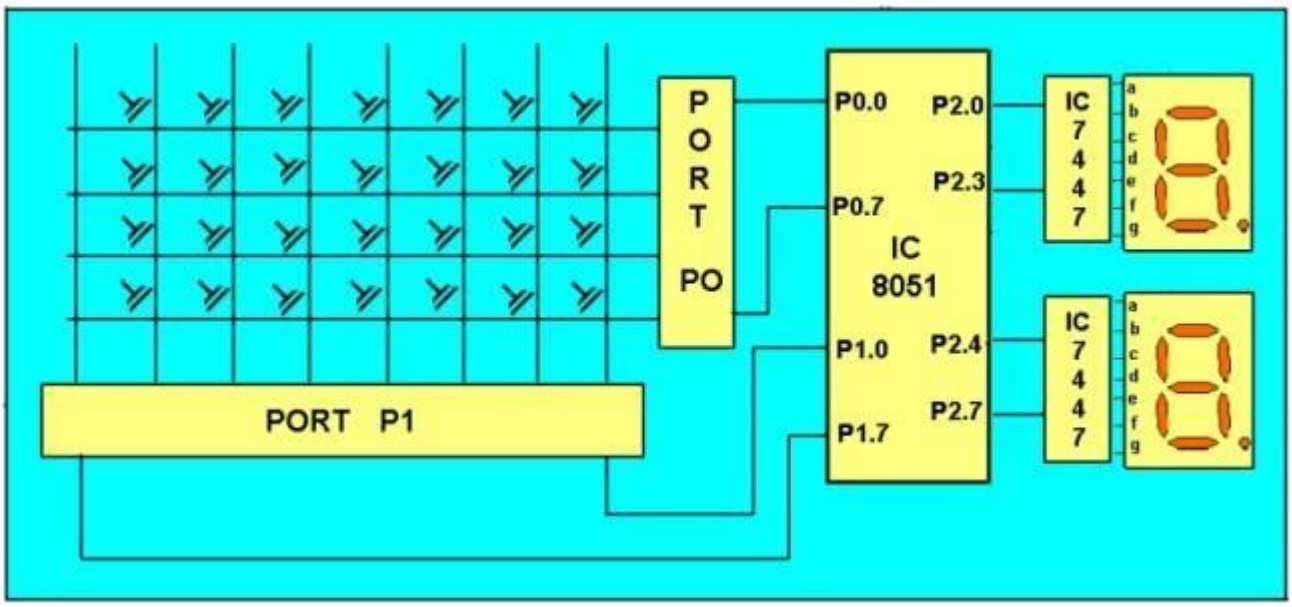
To check the row of the pressed key in the keyboard, one of the row is made high by making one of bit in the output port of 8051 high . This is done until the row is found out. Once we get the row next out job is to find out the column of the pressed key. The column is detected by contents in the input 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.

Circuit diagram of *INTERFACING KEY BOARD TO 8051.*

The programming algorithm, program and the circuit diagram is as follows. Here program is explained with comments.



Circuit diagram of *INTERFACING KEY BOARD TO 8051.*

Keyboard is organized in a matrix of rows and columns as shown in the figure. The microcontroller accesses both rows and columns through the port.

1. **The 8051 has 4 I/O ports P0 to P3 each with 8 I/O pins, P0.0 to P0.7,P1.0 to P1.7, P2.0 to P2.7, P3.0 to P3.7. The one of the port P1 (it understood that P1 means P1.0 to P1.7) as an I/P port for microcontroller 8051, port P0 as an O/P port of microcontroller 8051 and port P2 is used for displaying the number of pressed key.**
2. **Make all rows of port P0 high so that it gives high signal when key is pressed.**
3. **See if any key is pressed by scanning the port P1 by checking all columns for non zero condition.**
4. **If any key is pressed, to identify which key is pressed make one row high at a time.**
5. **Initiate a counter to hold the count so that each key is counted.**
6. **Check port P1 for nonzero condition. If any nonzero number is there in [accumulator], start column scanning by following step 9.**
7. **Otherwise make next row high in port P1.**
8. **Add a count of 08h to the counter to move to the next row by repeating steps from step 6.**
9. **If any key pressed is found, the [accumulator] content is rotated right through the carry until carry bit sets, while doing this increment the count in the counter till carry is found.**
10. **Move the content in the counter to display in data field or to memory location**
11. **To repeat the procedures go to step 2.**

Program to interface matrix keyboard to microcontroller 8051

Start of main program:

to check that whether any key is pressed

```
start:  mov a,#00h
        mov p1,a        ;making all rows of port p1 zero
        mov a,#0fh
        mov p1,a        ;making all rows of port p1 high
press:  mov a,p2
        jz press        ;check until any key is pressed
```

after making sure that any key is pressed

```
        mov a,#01h      ;make one row high at a time
        mov r4,a
        mov r3,#00h     ;initiating counter
next:   mov a,r4
        mov p1,a        ;making one row high at a time
        mov a,p2        ;taking input from port A
        jnz colscan     ;after getting the row jump to check
                          column
        mov a,r4
        rl a            ;rotate left to check next row
        mov r4,a
        mov a,r3
        add a,#08h      ;increment counter by 08 count
        mov r3,a
        sjmp next       ;jump to check next row
```

after identifying the row to check the colomn following steps are followed

```
colscan:  mov r5,#00h
in:       rrc a         ;rotate right with carry until get the carry
          jc out        ;jump on getting carry
          inc r3        ;increment one count
          jmp in
out:      mov a,r3
          da a          ;decimal adjust the contents of counter
                          before display
          mov p2,a
          jmp start     ;repeat for check next key.
```

3.11 Closed loop control of servo motor

servomotors / Servo Drivers

Related Contents

- **Servomotors / Servo Drivers**
  - Products

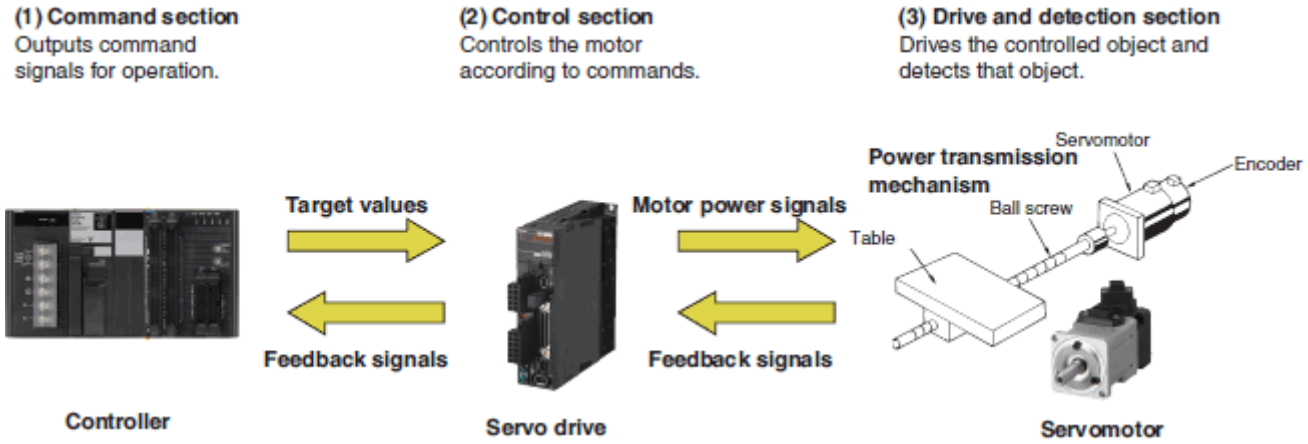
Primary Contents

- What Is a Servomotor and What Is a Servo Drive?
- Features
- Principles

What Is a Servomotor and What Is a Servo Drive?

A servomotor is a structural unit of a servo system and is used with a servo drive. The servomotor includes the motor that drives the load and a position detection component, such as an encoder. The servo system vary the controlled amount, such as position, speed, or torque, according to the set target value (command value) to precisely control the machine operation.

Servo System Configuration Example



- Top of page

Features

Precise, High-speed Control

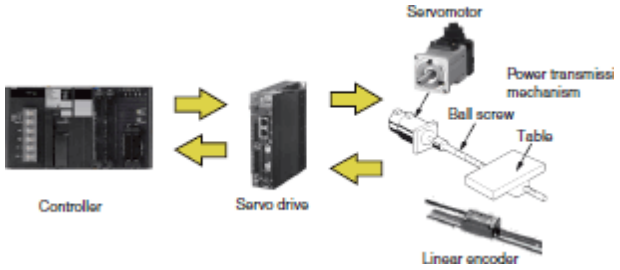
Servomotors excel at position and speed control.
Precise and flexible positioning is possible.

Servomotors do not stall even at high speeds. Deviations due to large external forces are corrected because encoders are used to monitor movement.

Fully-closed Loop

The most reliable form of closed loop. A fully-closed loop is used when high precision is required. The motor is controlled while directly reading the position of the machine (workpiece or table) using a linear encoder and comparing the read position with the command value (target value). Therefore, there is no need to compensate for gear backlash between the motor and mechanical system, feed screw pitch error, or error due to feed screw torsion or expansion.

Fully-closed Loop System Configuration Example



Semi-closed Loop

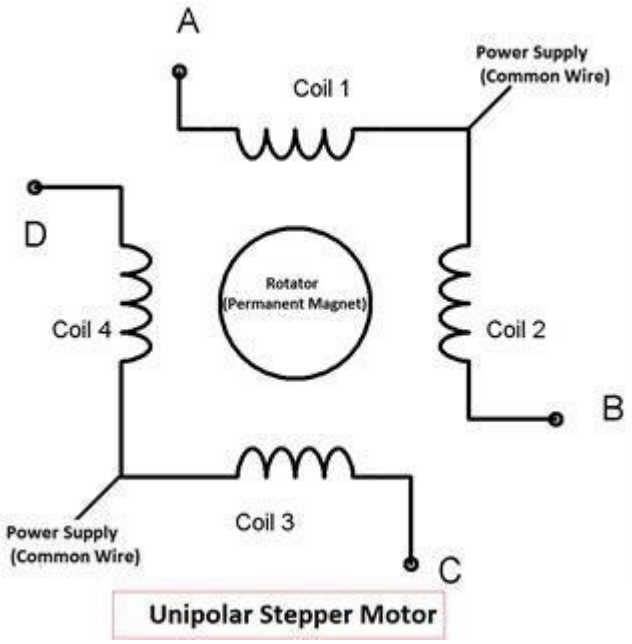
This method is commonly used in servo systems.
It is faster and has better positioning precision than an open loop.
Typically an encoder or other detector is attached behind the motor. The encoder detects the rotation angle of a feed screw (ball screw) and provides it as feedback of the machine (workpiece or table) travel position. This means that the position of the machine is not detected directly.
The characteristics depend on where the detector is installed.

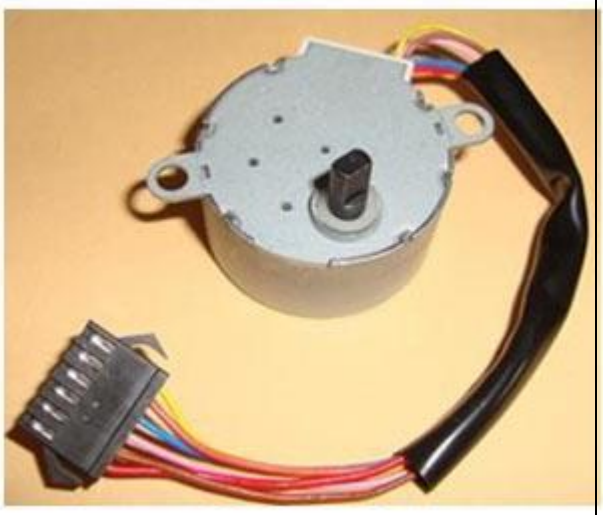
Installation location of detector	Behind motor	Motor side of feed screw	Opposite of motor side of feed screw
Gear backlash	Compensation required	Compensation not required	←
Ball screw or nut torsion	Affected	←	Hardly affected
Ball screw expansion or contraction	Affected	←	←
Ball screw pitch error	Compensation required	←	←

Semi-closed Loop System Configuration Example

3.12 stepper motor control

.





In Bipolar stepper motor there is just four wires coming out from two sets of coils, means there are no common wire.

Stepper motor is made up of a stator and a rotator. Stator represents the four electromagnet coils which remain stationary around the rotator, and rotator represents permanent magnet which rotates. Whenever the coils energised by applying the current, the electromagnetic field is created, resulting the rotation of rotator (permanent magnet). Coils should be energised in a particular sequence to make the rotator rotate. On the basis of this "sequence" we can divide the working method of Unipolar stepper motor in three modes: Wave drive mode, full step drive mode and half step drive mode.

Wave drive mode: In this mode one coil is energised at a time, all four coil are energised one after another. It produces less torque in compare with Full step drive mode but power consumption is less. Following is the table for producing this mode using microcontroller, means we need to give Logic 1 to the coils in the sequential manner

Steps	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Full Drive mode: In this, two coil are energised at the same time producing high torque. Power consumption is higher. We need to give Logic 1 to two coils at the same time, then to the next two coils and so on.

Steps	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

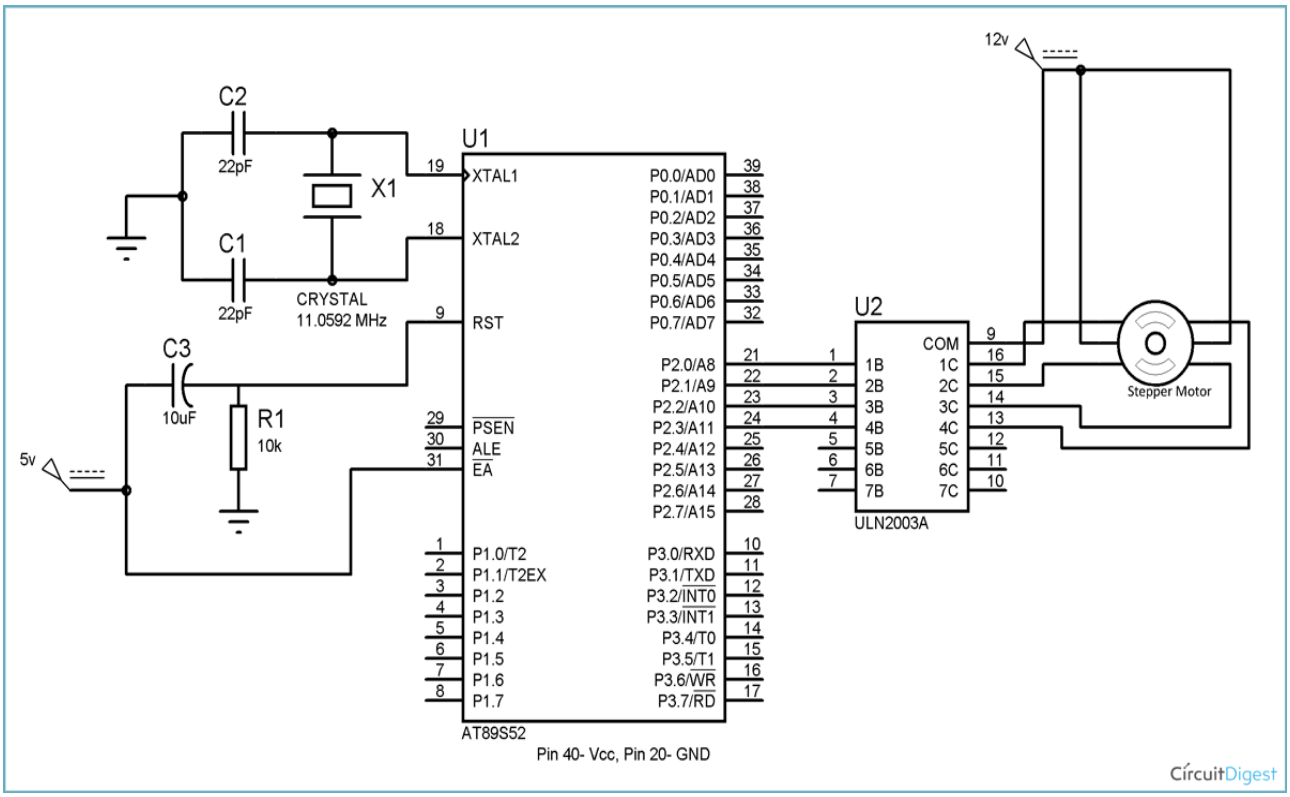
Half Drive mode: In this mode one and two coils are energised alternatively, means firstly one coil is energised then two coils are energised then again one coil is energised then again two, and so on. This is combination of full and wave drive mode, and used to increase the angular rotation of the motor.

Steps	A	B	C	D
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0

5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Interfacing Stepper Motor with 8051 Microcontroller

Interfacing with 8051 is very easy we just need to give the 0 and 1 to the four wires of stepper motor according to the above tables depending on which mode we want to run the stepper motor. And rest two wires should be connected to a proper 12v supply (depending on the stepper motor). Here we have used the unipolar stepper motor. We have connected four ends of the coils to the first four pins of port 2 of 8051 through the ULN2003A.



8051 doesn't provide enough current to drive the coils so we need to use a **current driver IC that is ULN2003A**. ULN2003A is the array of seven NPN Darlington transistor pairs. Darlington pair is constructed by connecting two bipolar transistors to achieve high current amplification. In ULN2003A, 7 pins are input pins and 7 pins are output pins, two pins are for Vcc (power supply) and Ground. Here we are using four input and four output pins. We can also use L293D IC in place of ULN2003A for current amplification.

You need to find out four coil wires and two common wires very carefully otherwise motor will not rotate. You can find it out by measuring resistance using multimeter, multimeter won't show any readings between the wires of two phases. Common wire and the other two wire in the same phase should show the same resistance, and the two end points of the two coils in the same phase will show the twice resistance in compared with resistance between common point and one end point.

Two mark questions and Answers

1. What are the special functoin register?

The special function register are stack pointer, index pointer (DPL and DPH), I/O port addresses, status(PSW) and accumulator.

2. What are the uses of accumulator register?

The accumulator registers (A and B at addresses OEOh and OFOh, respectively) are used to store temporary values and the results of arithmetic operations.

3. What is PSW?

Program status word (PSW) is the set of flags that contains the status information and is considered as one of the special function register.

4. What is stack pointer (sp)?

Stack pointer (SP) is a 8 bit wide register and is incremented before the data is stored into the stack using PUSH or CALL instructions.

It contains 8-bit stack top address. It is defined anywhere in the on-chip 128-byte RAM. After reset, the SP register is initialized to 07.

After each write to stack operation, the 8-bit contents of the operand are stored onto the stack, after incrementing the SP register by one.

It is not a top-down data structure. It is allotted an address in the special function register bank.

5. What is data pointer (DTPR)?

It is a 16-bit register that contains a higher byte (DPH) and lower byte (DPL) of a 16-bit external data RAM address.

It is accessed as a 16-bit register or two 8-bit registers. It has been allotted two addresses in the special function register bank, for its two bytes DPH and DPL.

6. Why oscillator circuit is used?

Oscillator circuit is used to generate the basic timing clock signal for the operation of the circuit using crystal oscillator.

7. What is the purpose of using instruction register?

Instruction register is used for the purpose of decoding the opcode of an instruction to be executed and gives information to the timing and control unit generating necessary signals for the execution of the instruction.

8. Give the purpose of ale/prog signal.

ALE/PROG is an address latch enable output pulse and indicates that valid address bits available on the respective pins.

The ALE pulses are emitted at a rate of one-sixth of the oscillator frequency. The signal is valid only for external memory accesses.

It may be used for external timing or clockwise purpose. One ALE pulse is skipped during each access to external data memory.

9. Explain the two power saving mode of operation. The two power saving modes of operation are:

I. Idle mode:

In this mode, the oscillator continues to run and the interrupt, serial port and timer blocks are active, but the clock to the CPU is disabled. The CPU status is preserved. This mode can be terminated with a hardware interrupt or hardware reset signal. After this, the CPU resumes program execution from where it left off.

II. Power down mode:

In this mode, the on-chip oscillator is stopped. All the functions of the controller are held maintaining the contents of RAM. The only way to terminate this mode is hardware reset. The reset redefines all the SFRs but the RAM contents are left unchanged.

10. Differentiate between program memory and data memory. i. In stores the programs to be executed.

ii. It stores only program code which is to be executed and thus it need not be written, so it is implemented using EPROM It stores the data, line intermediate results, variables and constants required for the execution of the program.

The data memory may be read from or written to and thus it is implemented using RAM.

11. What are addressing modes?

The various ways of accessing data are called addressing modes.

12. Give the addressing modes of 8051?

There are six addressing modes in 8051.They are Direct addressing Indirect addressing Register instruction

Registerspecific (register implicit)

Immediate mode Indexed addressing

10 mark questions

1. List all the registers used in 8051 microcontroller in brief.
2. Draw the pin diagram of 8051 microcontroller and explain each one.
3. Draw the memory organization of micro controller 8051 and eplain.
4. What are all addressing modes of micro controller 8051? explain with examples.
5. Draw the block diagram of 8051 microcontroller and explain.
6. What is 8051 micro-controller system? ? Give its block diagram representation? Also describe its all ports in details?

UNIT 4

MOTOR CONTROL SIGNAL PROCESSORS

4.1 INTRODUCTION

The Texas Instruments TMS320LF2407 DSP Controller (referred to as the LF2407 in this text) is a programmable digital controller with a C2xx DSP central processing unit (CPU) as the core processor. The LF2407 contains the DSP core processor and useful peripherals integrated onto a single piece of silicon. The LF2407 combines the powerful CPU with on-chip memory and peripherals. With the DSP core and control-oriented peripherals integrated into a single chip, users can design very compact and cost-effective digital control systems.

The LF2407 DSP controller offers 40 million instructions per second (MIPS) performance. This high processing speed of the C2xx CPU allows users to compute parameters in real time rather than look up approximations from tables stored in memory. This fast performance is well suited for processing control parameters in applications such as notch filters or sensor less motor control algorithms where a large amount of calculations must be computed quickly.

While the "brain" of the LF2407 DSP is the C2xx core, the LF2407 contains several control-orientated peripherals onboard (see Fig. 3.1). The peripherals on the LF2407 make virtually any digital control requirement possible. Their applications range from analog to digital conversion to pulse width modulation (PWM) generation. Communication peripherals make possible the communication with external peripherals, personal computers, or other DSP processors. Below is a brief listing of the different peripherals onboard the LF2407 followed by a graphical layout depicted in Fig. 3.1.

The LF2407 peripheral set includes:

- Two Event Managers (A and B)
- General Purpose (GP) timers
- PWM generators for digital motor control
- Analog-to-digital converter
- Controller Area Network (CAN) interface

- Serial Peripheral Interface (SPI) – synchronous serial port
- Serial Communications Interface (SCI) – asynchronous serial port
- General-Purpose bi-directional digital I/O (GPIO) pins
- Watchdog Timer ("time-out" DSP reset device for system integrity)

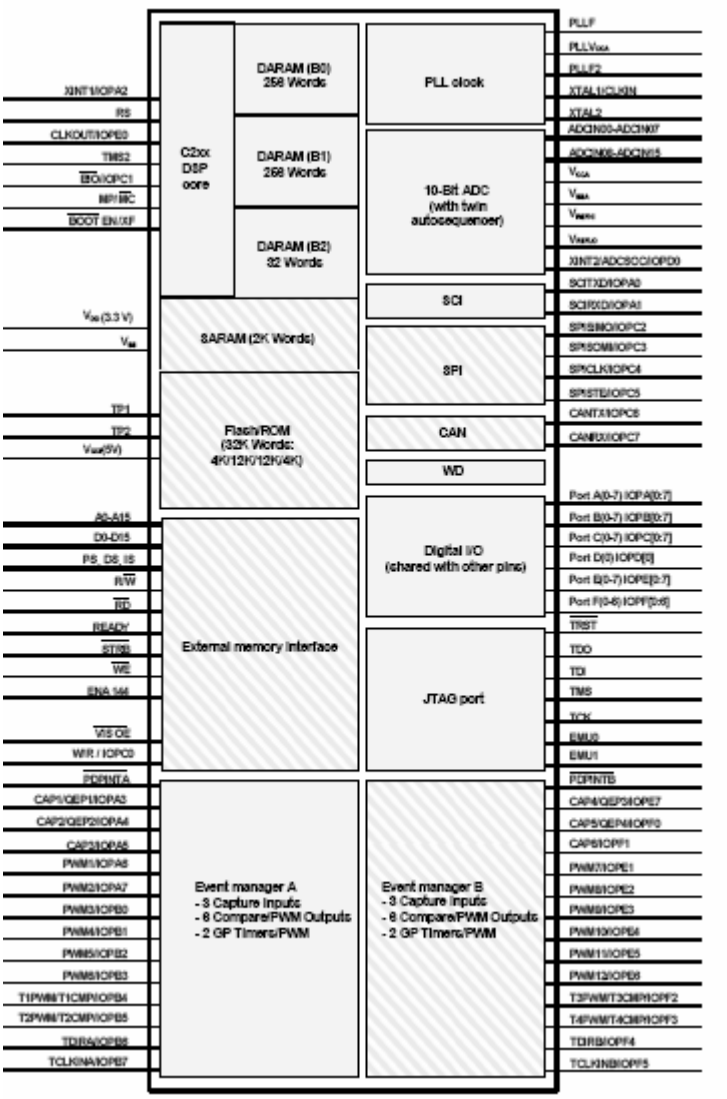


Figure 3.1 Graphical overview of DSP core and peripherals on the LF2407.

4.2 Brief Introduction to Peripherals

The following peripherals are those that are integrated onto the LF2407 chip. Refer to Fig. 1.1 to view the pin-out associated with each peripheral.

Event Managers (EVA, EVB)

There are two Event Managers on the LF2407, the EVA and EVB. The Event Manager is the most important peripheral in digital motor control. It contains the necessary functions needed to control electromechanical devices. Each EV is composed of functional "blocks" including timers, comparators, and capture units for triggering on an event, PWM logic circuits, quadrature-encoder–pulse (QEP) circuits, and interrupt logic.

The Analog-to-Digital Converter (ADC)

The ADC on the LF2407 is used whenever an external analog signal needs to be sampled and converted to a digital number. Examples of ADC applications range from sampling a control signal for use in a digital notch filtering algorithm or using the ADC in a control feedback loop to monitor motor performance. Additionally, the ADC is useful in motor control applications because it allows for current sensing using a shunt resistor instead of an expensive current sensor.

The Control Area Network (CAN) Module

While the CAN module will not be covered in this text, it is a useful peripheral for specific applications of the LF2407. The CAN module is used for multi-master serial communication between external hardware. The CAN bus has a high level of data integrity and is ideal for operation in noisy environments such as in an automobile, or industrial environments that require reliable communication and data integrity.

Serial Peripheral Interface (SPI) and Serial Communications Interface (SCI)

The SPI is a high-speed synchronous communication port that is mainly used for communicating between the DSP and external peripherals or another DSP device. Typical uses of the SPI include communication with external shift registers, display drivers, or ADCs.

The SCI is an asynchronous communication port that supports asynchronous serial (UART) digital communication between the CPU and other asynchronous peripherals that use the standard NRZ (non-return-to-zero) format. It is useful in communication between external devices and the DSP. Since these communication peripherals are not directly related to motion control applications, they will not be discussed further in this text.

Watchdog Timer (WD)

The Watchdog timer (WD) peripheral monitors software and hardware operations and asserts a system reset when its internal counter overflows. The WD timer (when enabled) will count for a specific amount of time. It is necessary for the user's software to reset the WD timer periodically so that an unwanted reset does not occur. If for some reason there is a CPU disruption, the watchdog will generate a system reset. For example, if the software enters an endless loop or if the CPU becomes temporarily disrupted, the WD timer will overflow and a DSP reset will occur, which will cause the DSP program to branch to its initial starting point. Most error conditions that temporarily disrupt chip operation and inhibit proper CPU function can be cleared by the WD function. In this way, the WD increases the reliability of the CPU, thus ensuring system integrity.

General Purpose Bi-Directional Digital I/O (GPIO) Pins

Since there are only a finite number of pins available on the LF2407 device, many of the pins are multiplexed to either their primary function or the secondary GPIO function. In most cases, a pin's second function will be as a general-purpose input/output pin. The GPIO capability of the LF2407 is very useful as a means of controlling the functionality of pins and also provides another method to input or output data to and from the device. Nine 16-bit control registers control all I/O and shared pins. There are two types of these registers:

- I/O MUX Control Registers (MCRx) – Used to control the multiplexer selection that Chooses between the primary function of a pin or the general-purpose I/O function.
- Data and Direction Control Registers (PxDATDIR) – Used to control the data and data Direction of bi-directional I/O pins.

Phase Locked Loop (PLL) Clock Module

The phase locked loop (PLL) module is basically an input clock multiplier that allows the user to control the input clocking frequency to the DSP core. External to the LF2407, a clock reference (can oscillator/crystal) is generated. This signal is fed into the LF2407 and is multiplied or divided by the PLL. This new (higher or lower frequency) clock signal is then used to clock the DSP core. The LF2407's PLL allows the user to select a multiplication factor ranging from 0.5X to 4X that of the external clock signal. The default value of the PLL is 4X.

Memory Allocation Spaces

The LF2407 DSP Controller has three different allocations of memory it can use: Data, Program, and I/O memory space. Data space is used for program calculations, look-up tables, and any other memory used by an algorithm. Data memory can be in the form of the on-chip random access memory (RAM) or external RAM. Program memory is the location of user's program code. Program memory on the LF2407 is either mapped to the off-chip RAM (MP/MC- pin =1) or to the on-chip flash memory (MP/MC- = 0), depending on the logic value of the MP/MC-pin.

I/O space is not really memory but a virtual memory address used to output data to peripherals external to the LF2407. For example, the digital-to-analog converter (DAC) on the Spectrum Digital™ evaluation module is accessed with I/O memory. If one desires to output data to the DAC, the data is simply sent to the configured address of I/O space with the "OUT" command. This process is similar to writing to data memory except that the OUT command is used and the data is copied to and outputted on the DAC instead of being stored in memory.

4.3. Types of Physical Memory

Random Access Memory (RAM)

The LF2407 has 544 words of 16 bits each in the on-chip DARAM. These 544 words are partitioned into three blocks: B0, B1, and B2. Blocks B1 and B2 are allocated for use only as data memory. Memory block B0 is different than B1 and B2. This memory block is normally configured as Data Memory, and hence primarily used to hold data, but in the case of the B0 block, it can also be configured as Program Memory. B0 memory can be configured as program or data memory depending on the value of the core level "CNF" bit.

- (CNF=0) maps B0 to data memory.
- (CNF=1) maps B0 to program memory.

The LF2407 also has 2K of single-access RAM (SARAM). The addresses associated with the SARAM can be used for both data memory and program memory, and are software configurable to the internal SARAM or external memory.

Non-Volatile Flash Memory

The LF2407 contains 32K of on-chip flash memory that can be mapped to program space if the MP/MC-pin is made logic 0 (tied to ground). The flash memory provides a permanent location to store code that is unaffected by cutting power to the device. The flash memory can be electronically programmed and erased many times to allow for code development. Usually, the external RAM on the LF2407 Evaluation Module (EVM) board is used instead of the flash for code development due to the fact that a separate "flash programming" routine must be performed to flash code into the flash memory. The on-chip flash is normally used in situations where the DSP program needs to be tested where a JTAG connection is not practical or where the DSP needs to be tested as a "stand-alone" device. For example, if a LF2407 was used to develop a DSP control solution to an automobile braking system, it would be somewhat impractical to have a DSP/JTAG/PC interface in a car that is undergoing performance testing.

4.1.1 Introduction to the C2xx DSP Core and Code Generation

The heart of the LF2407 DSP Controller is the C2xx DSP core. This core is a 16-bit fixed point processor, meaning that it works with 16-bit binary numbers. One can think of the C2xx as the central processor in a personal computer. The LF2407 DSP consists of the C2xx DSP core plus many peripherals such as Event Managers, ADC, etc., all integrated onto one single chip.

4.1.2. The Components of the C2xx DSP Core

The DSP core (like all microprocessors) consists of several subcomponents necessary to perform arithmetic operations on 16-bit binary numbers. The following is a list of the multiple subcomponents found in the C2xx core which we will discuss further:

- A 32-bit central arithmetic logic unit (CALU)
- A 32-bit accumulator (used frequently in programs)
- Input and output data-scaling shifters for the CALU
- A (16-bit by 16-bit) multiplier
- A product-scaling shifter
- Eight auxiliary registers (AR0 – AR7) and an auxiliary register arithmetic unit (ARAU)

Each of the above components is either accessed directly by the user code or is indirectly used during the execution of an assembly command.

Central Arithmetic Logic Unit (CALU)

The C2xx performs 2s-complement arithmetic using the 32-bit CALU. The CALU uses 16-bit words taken from data memory, derived from an immediate instruction, or from the 32-bit multiplier result. In addition to arithmetic operations, the CALU can perform Boolean operations. The CALU is somewhat transparent to the user. For example, if an arithmetic command is used, the user only needs to write the command and later read the output from the appropriate register. In this sense, the CALU is "transparent" in that it is not accessed directly by the user.

Accumulator

The accumulator stores the output from the CALU and also serves as another input to the CALU (many arithmetic commands perform operations on numbers that are currently stored in the accumulator; versus other memory locations). The accumulator is 32 bits wide and is divided into two sections, each consisting of 16 bits. The high-order bits consist of bits 31 through 16, and the low-order bits are made up of bits 15 through 0. Assembly language instructions are provided for storing the high- and low-order accumulator words to data memory. In most cases, the accumulator is written to and read from directly by the user code via assembly commands. In some instances, the accumulator is also transparent to the user (similar to the CALU operation in that it is accessed "behind the scenes").

Scaling Shifters

The C2xx has three 32-bit shifters that allow for scaling, bit extraction, extended arithmetic, and overflow-prevention operations. The scaling shifters make possible commands that shift data left or right. Like the CALU, the operation of the scaling shifters is "transparent" to the user. For example, the user needs only to use a shift command, and observe the result. Any one of the three shifters could be used by the C2xx depending on the specific instruction entered. The following is a description of the three shifters:

- Input data-scaling shifter (input shifter): This shifter left-shifts 16-bit input data by 0 to 16 bits to align the data to the 32-bit input of the CALU. For example, when the user uses a command such as "ADD 300h, 5", the input shifter is responsible for first shifting the data in memory address "300h" to the left by five places before it is added to the contents of the accumulator.

- Output data-scaling shifter (output shifter): This shifter left-shifts data from the accumulator by 0 to 7 bits before the output is stored to data memory. The content of the accumulator remains unchanged. For example, when the user uses a command such as "SACL 300h, 4", the output shifter is responsible for first shifting the contents of the accumulator to the left by four places before it is stored to the memory address "300h".
- Product-scaling shifter (product shifter): The product register (PREG) receives the output of the multiplier. The product shifter shifts the output of the PREG before that output is sent to the input of the CALU. The product shifter has four product shift modes (no shift, left shift by one bit, left shift by four bits, and right shift by six bits), which are useful for performing multiply/accumulate operations, fractional arithmetic, or justifying fractional products.

Multiplier

The multiplier performs 16-bit, 2s-complement multiplication and creates a 32-bit result. In conjunction with the multiplier, the C2xx uses the 16-bit temporary register (TREG) and the 32-bit product register (PREG).

The operation of the multiplier is not as "transparent" as the CALU or shifters. The TREG always needs to be loaded with one of the numbers that are to be multiplied. Other than this prerequisite, the multiplication commands do not require any more actions from the user code. The output of the multiply is stored in the PREG, which can later be read by the user code.

Auxiliary Register Arithmetic Unit (ARAU) and Auxiliary Registers

The ARAU generates data memory addresses when an instruction uses indirect addressing to access data memory (more on indirect addressing will be covered later along with assembly programming). Eight auxiliary registers (AR0 through AR7) support the ARAU, each of which can be loaded with a 16-bit value from data memory or directly from an instruction. Each auxiliary register value can also be stored in data memory. The auxiliary registers are mainly used as "pointers" to data memory locations to more easily facilitate looping or repeating algorithms. They are directly written to by the user code and are automatically incremented or decremented by particular assembly instructions during a looping or repeating operation. The auxiliary register pointer (ARP) embedded in status register ST0 references the auxiliary register. The status registers (ST0, ST1) are core level registers where values such as the Data Page (DP) and ARP located.

4.2. System Configuration Registers

The System Control and Status Registers (SCSR1, SCSR2) are used to configure or display fundamental settings of the LF2407. For example, these fundamental settings include the clock speed (clock pre-scale setting) of the LF2407, which peripherals are enabled, microprocessor/microcontroller mode, etc. Bits are controlled by writing to the corresponding data memory address or the logic level on an external pin as with the microprocessor/microcontroller (MP/MC) select bit. The bit descriptions of these two registers (mapped to data memory) are listed below.

System Control and Status Register 1 (SCSR1) — Address 07018h

15	14	13	12	11	10	9	8
Reserved	CLKSRC	LPM1	LPM0	CLK PS2	CLK PS1	CLK PS0	Reserved
R–0	RW–0	RW–0	RW–0	RW–1	RW–1	RW–1	R–0
7	6	5	4	3	2	1	0
ADC CLKEN	SCI CLKEN	SPI CLKEN	CAN CLKEN	EVB CLKEN	EVA CLKEN	Reserved	ILLADR
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	R–0	RC–0

Note: *R = read access, W = write access, C = clear, -0 = value after reset.*

Bit 15 Reserved

Bit 14 CLKSRC. CLKOUT pin source select

0 CLKOUT pin has CPU Clock (40 MHz on a 40-MHz device) as the output

1 CLKOUT pin has Watchdog clock as the output

Bits 13–12 LPM (1:0). Low-power mode select

These bits indicate which low-power mode is entered when the CPU executes the IDLE instruction. Description of the low-power modes:

LPM(1:0)	Low-Power mode selected
00	IDLE1 (LPM0)
01	IDLE2. (LPM1)
1x	HALT (LPM2)

Bits 11–9

PLL Clock prescale select. These bits select the PLL multiplication factor for the input clock.

CLK PS2	CLK PS1	CLK PS0	System Clock Frequency
0	0	0	$4 \times F_{in}$
0	0	1	$2 \times F_{in}$
0	1	0	$1.33 \times F_{in}$
0	1	1	$1 \times F_{in}$
1	0	0	$0.8 \times F_{in}$
1	0	1	$0.66 \times F_{in}$
1	1	0	$0.57 \times F_{in}$
1	1	1	$0.5 \times F_{in}$

Note: F_{in} is the input clock frequency.

Bit 8 Reserved

Bit 7 ADC CLKEN. ADC module clock enable control bit.

0 Clock to module is disabled (i.e., shut down to conserve power).

1 Clock to module is enabled and running normally.

Bit 6 SCI CLKEN. SCI module clock enable control bit.

0 Clock to module is disabled (i.e., shut down to conserve power).

1 Clock to module is enabled and running normally.

Bit 5 SPI CLKEN. SPI module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 4 CAN CLKEN. CAN module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 3 EVB CLKEN. EVB module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 2 EVA CLKEN. EVA module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Note: In order to modify/read the register contents of any peripheral, the clock to that peripheral must be enabled by writing a 1 to the appropriate bit.

Bit 1 Reserved

Bit 0 ILLADR. Illegal Address detect bit

If an illegal address has occurred, this bit will be set. It is up to software to clear this bit following an illegal address detects. This bit is cleared by writing a 1 to it and should be cleared as part of the initialization sequence. Note: An illegal address will cause a Non-Mask able Interrupt (NMI).

System Control and Status Register 2 (SCSR2) — Address 07019h

15-8							
Reserved							
RW–0							
7	6	5	4	3	2	1	0
Reserved	I/P QUAL	WD OVERRIDE	XMIF HI–Z	BOOT EN	MP/MC	DON	PON
	RW–0	RC–1	RW–0	RW–BOOT EN pin	RW–MP/MC pin	RW–1	RW–1

Note: *R = read access, W = write access, C = clear, -0 = value after reset.*

Bits 15–7 Reserved. Writes have no effect; reads are undefined

Bit 6 Input Qualifier Clocks.

An input-qualifier circuitry qualifies the input signal to the CAP1–6, XINT1/2, ADCSOC, and PDPINTA/B pins in the 240xA devices. The I/O functions of these pins do not use the input-qualifier circuitry. The state of the internal input signal will change only after the pin is held high/low for 6 (or 12) clock edges. This ensures that a glitch smaller than (or equal to) 5 (or 11) CLKOUT cycles wide will not change the internal pin input state. The user must hold the pin high/low for 6 (or 12) cycles to ensure that the device will see the level change. This bit determines the width of the glitches (in number of internal clock cycles) that will be blocked. Note that the internal clock is not the same as CLKOUT, although its frequency is the same as CLKOUT.

0 The input-qualifier circuitry blocks glitches up to 5 clock cycles long

1 The input-qualifier circuitry blocks glitches up to 11 clock cycles long

Note: This bit is applicable only for the 240xA devices, not for the 240x devices because they lack an input-qualifier circuitry.

Bit 5 Watchdog Override. (WD protect bit)

After RESET, this bit gives the user the ability to disable the WD function through software (by setting the WDDIS bit = 1 in the WDCR). This bit is a clear-only bit and defaults to a 1 after reset.

Note: This bit is cleared by writing a 1 to it.

0 Protects the WD from being disabled by software. This bit cannot be set to 1 by software. It is a clear-only bit, cleared by writing a 1.

1 This is the default reset value and allows the user to disable the WD through the WDDIS bit in the WDCR. Once cleared, however, this bit can no longer be set to 1 by software, thereby protecting the integrity of the WD timer.

Bit 4 XMIF Hi-Z Control

This bit controls the state of the external memory interface (XMIF) signals.

0 XMIF signals in normal driven mode; i.e., not Hi-Z (high impedance).

1 All XMIF signals are forced to Hi-Z state.

Bit 3 Boot Enable

This bit reflects the state of the BOOT_EN / XF pin at the time of reset. After reset and device has "booted up", this bit can be changed in software to re-enable Flash memory visibility or return to active Boot ROM.

0 Enable Boot ROM — Address space 0000 — 00FF is now occupied by the on-chip Boot ROM Block. Flash memory is totally disabled in this mode. Note: There is no on-chip boot ROM in ROM devices (i.e., LC240xA)

1 Disable Boot ROM — Program address space 0000 — 7FFF is mapped to on-chip Flash memory in the case of LF2407A and LF2406A. In the case of LF2402A, addresses 0000 – 1FFF are mapped

Bit 2 Microprocessor/Microcontroller Select

This bit reflects the state of the MP/MC pin at time of reset. After reset, this bit can be changed in software to allow dynamic mapping of memory on and off chip.

0 Set to Microcontroller mode — Program Address range 0000 — 7FFF is mapped internally (i.e., Flash)

1 Set to Microprocessor mode — Program Address range 0000 — 7FFF is mapped externally (i.e., customer provides external memory device.)

Bits 1–0 SARAM Program/Data Space Select

DON	PON	SARAM status
0	0	SARAM not mapped (disabled), address space allocated to external memory
0	1	SARAM mapped internally to Program space
1	0	SARAM mapped internally to Data space
1	1	SARAM block mapped internally to both Data and Program spaces.

This is the default or reset value

4.3. Memory Addressing Modes

There are three basic memory addressing modes used by the C2xx instruction set. The three modes are:

- Immediate addressing mode (does not actually access memory)
- Direct addressing mode
- Indirect addressing mode

4.4.1 Immediate Addressing Mode

In the immediate addressing mode, the instruction contains a constant to be manipulated by the instruction. Even though the name "immediate addressing" suggests that a memory location is accessed, immediate addressing is simply dealing with a user-specified constant which is usually included in the assembly command syntax. The "#" sign indicates that the value is an immediate address (just a constant). The two types of immediate addressing modes are:

Short-immediate addressing. The instructions that use short-immediate addressing have an 8-bit, 9-bit, or 13-bit constant as the operand.

For example, the instruction:

```
LACL #44h          ; loads lower bits of accumulator with
                   ; Eight-bit constant (44h in this case)
```

Note: The LACL command will work only with a short 8-bit constant. If you want to load a long 16-bit constant, then use the LACC command.

Long-immediate addressing. Instructions that use long-immediate addressing have a 16-bit constant as an operand. This 16-bit value can be used as an absolute constant or as a 2s-complement value.

For example, the instruction:

```
LACC #4444h          ; loads accumulator with up to a 16-bit
```

; Constant (4444h in this case)

If you need to use registers or access locations in data memory, you must use either direct or indirect addressing.

4,5. Direct Addressing Mode

In direct addressing, data memory is first addressed in blocks of 128 words called data pages. The entire 64K of data memory consists of 512 DPs labeled 0 through 511, as shown in the Fig. 3.2. The current DP is determined by the value in the 9-bit DP pointer in status register ST0. For example, if the DP value is "0 0000 0000", the current DP is 0. If the DP value is "0 0000 0010", the current data page is 2. The DP of a particular memory address can be found easily by dividing the address (in hexadecimal) by 80h.

For example: For the data memory address 0300h, 300h/80h = 6h so the DP pointer is 6h. Likewise, the DP pointer for 200h is 4h.

DP Value	Offset	Data Memory
0000 0000 0 : 0000 0000 0	000 0000 : 111 1111	Page 0: 0000h-007Fh
0000 0000 1 : 0000 0000 1	000 0000 : 111 1111	Page 1: 0080h-00FFh
0000 0001 0 : 0000 0001 0	000 0000 : 111 1111	Page 2: 0100h-017Fh
. . . . . .	. . . . . .	. . . . . .
1111 1111 1 : 1111 1111 1	000 0000 : 111 1111	Page 511: FF80h-FFFFh

Figure 3.2 Data pages and corresponding memory ranges.

In addition to the DP, the DSP must know the particular word being referenced on that page. This is determined by a 7-bit offset. The 7-bit offset is simply the 7 least significant bits (LSBs) of the memory address. The DP and the offset make up the 16-bit memory address (see Fig. 3.3).

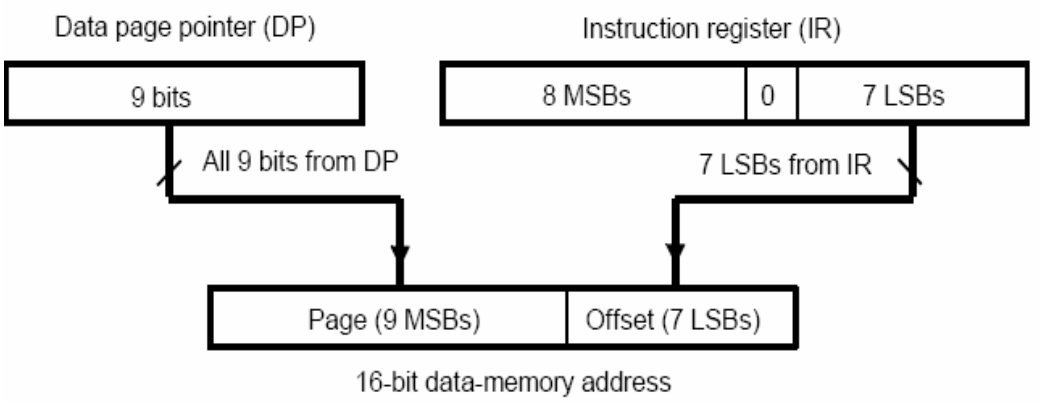


Figure 3.3 Data page and offset make up a 16-bit memory address.

When you use direct addressing, the processor uses the 9 DP bits and the 7 LSBs of the instruction to obtain the true memory address. The following steps should be followed when using direct addressing:

1 Set the DP. Load the appropriate value (from 0 to 511 in decimal or 0-1FF in hex) into the DP. The easiest way to do this is with the LDP instruction. The LDP instruction loads the DP directly to the ST0 register without affecting any other bits of the ST0.

```
LDP #0E1h          ; sets the data page pointer to E1h
                            Or
LDP #225           ; sets the data page pointer to 225 decimal
                   ; Which is E1 in hexadecimal
```

2 Specify the offset. For example, if you want the ADD instruction to use the value at the second address of the current data page, you would write: ADD 1h

If the data page points to 300h, then the above instruction will add the contents of 301h to the accumulator

Note: You do not have to set the data page prior to every instruction that uses direct addressing. If all the instructions in a block of code access the same data page, you can simply load the DP before the block. However, if various data pages are being accessed throughout the block of code be sure the DP is changed accordingly.

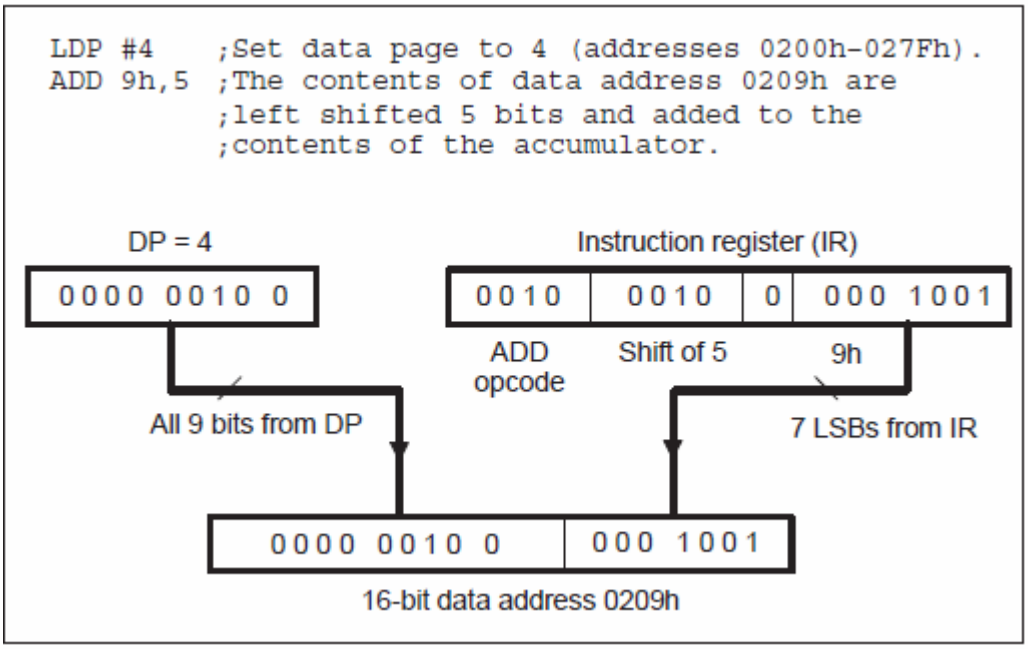
Examples of Direct Addressing

In Example 1, the first instruction loads the DP with $0\ 0000\ 0100_2$ to set the current data page to 4. The ADD instruction then references a data memory address that is generated as

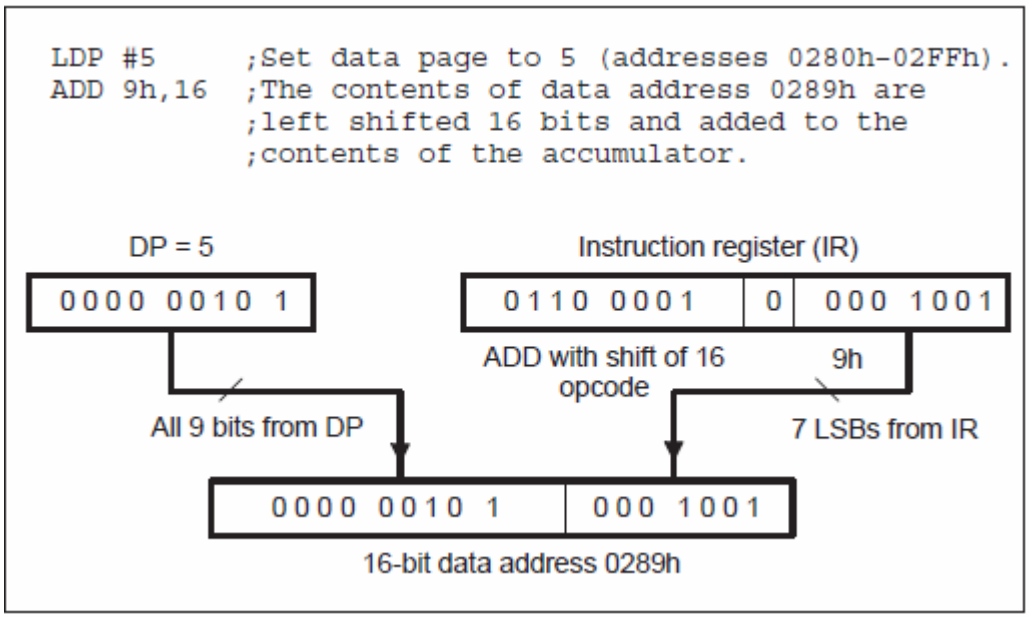
shown following the program code. Before the ADD instruction is executed, the opcode is loaded into the instruction register. Together, the DP and the seven LSBs of the instruction register form the complete 16-bit address, 0000 0010 0000 1001_2 (0209h).

In Example 2, the ADD instruction references a data memory address that is generated as shown following the program code. For any instruction that performs a shift of 16, the shift value is not embedded directly in the instruction word; instead, all eight MSBs contain an opcode that not only indicates the instruction type, but also a shift of 16. The eight MSBs of the instruction word indicate an ADD with a shift of 16.

Example 1 Using Direct Addressing with ADD (Shift of 0 to 15)

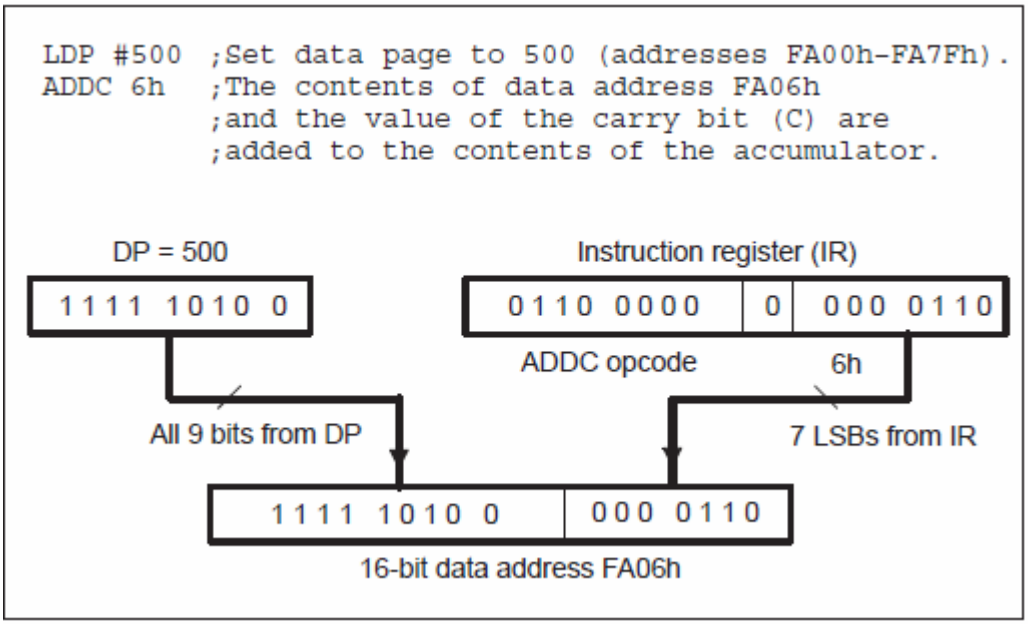


Example 2 Using Direct Addressing with ADD (Shift of 16)



In Example 3, the ADDC instruction references a data memory address that is generated as shown following the program code. You should note that if an instruction does not perform shifts (such as the ADDC instruction), all eight MSBs of the instruction contain the opcode for the instruction type.

Example 3 Using Direct Addressing with ADDC



4.5.2 Indirect Addressing Mode

Indirect addressing is a powerful way of addressing data memory. Indirect addressing mode is not dependent on the current data page as is direct addressing. Instead, when using indirect addressing you load the memory space that you would like to access into one of the auxiliary registers (ARx). The current auxiliary register acts as a pointer that points to a specific memory address.

The register pointed to by the ARP is referred to as the current auxiliary register or current AR. To select a specific auxiliary register, load the 3-bit auxiliary register pointer (ARP) with a value from 0 to 7. The ARP can be loaded with the MAR instruction or by the LARP instruction. An ARP value can also be loaded by using the ARx operand after any instruction that supports indirect addressing as seen below.

Example of using MAR:

```
ADD *, AR1          ; Adds using current *, then makes AR1 the
                    ; New current AR for future uses
```

Example of using LARP

```
LARP #2             ; this will make AR2 the current AR
```

The C2xx provides four types of indirect addressing options:

- No increment or decrement. The instruction uses the content of the current auxiliary register as the data memory addresses but neither increments nor decrements the content of the current auxiliary register.
- Increment or decrement by 1. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by one.
- Increment or decrement by an index amount. The value in AR0 is the index amount. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by the index amount.
- Increment or decrement by an index amount using reverse carry. The value in AR0 i the index amount. After the instruction uses the content of the current auxiliary register as the data memory address, that content is incremented or decremented by the index

amount. The addition and subtraction process is accomplished with the carry propagation reversed and is useful in fast Fourier transforms algorithms.

Table 4.1 displays the various operands that are available for use with instructions while using indirect addressing mode.

Operand	Option	Example
*	No increment or decrement	**LT** * loads the temporary register TREG with the content of the data memory address referenced by the current AR.
*+	Increment by 1	**LT** *+ loads the TREG with the content of the data memory address referenced by the current AR and then adds 1 to the content of the current AR.
*-	Decrement by 1	**LT** *- loads the TREG with the content of the data memory address referenced by the current AR and then subtracts 1 from the content of the current AR.
*0+	Increment by index amount	**LT** *0+ loads the TREG with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR.
*0-	Decrement by index amount	**LT** *0- loads the TREG with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR.
*BR0+	Increment by index amount, adding with reverse carry	**LT** *BR0+ loads the TREG with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR, adding with reverse carry propagation.
*BR0-	Decrement by index amount, subtracting with reverse carry	**LT** *BR0- loads the TREG with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR, subtracting with bit reverse carry propagation.

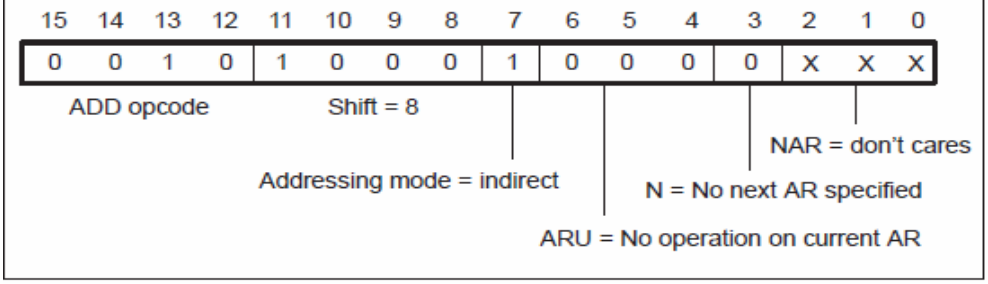
Table 3.1 Indirect addressing operands.

Examples of Indirect Addressing

Example 1 illustrates how the instruction register is loaded with the value shown when the ADD instruction is fetched from program memory.

Example 1. Indirect Addressing—No Increment or Decrement

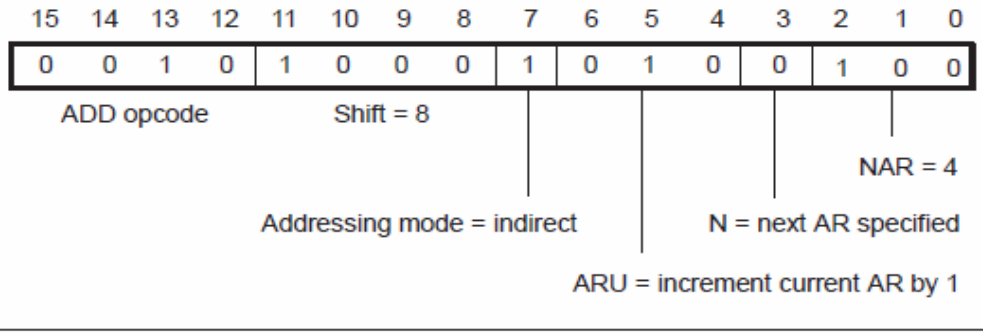
```
ADD *,8      ;Add to the accumulator the content of the
             ;data-memory address referenced by the
             ;current auxiliary register. The data
             ;is left shifted 8 bits before being added.
```



Example 2, illustrates how the instruction register is loaded with the value shown when the ADD instruction is fetched from program memory.

Example 2. Indirect Addressing—Increment by 1

```
ADD *+,8,AR4     ;Operates as in Example 1 , but
                 ;in addition, the current auxiliary
                 ;register is incremented by one, and
                 ;AR4 is chosen as the next auxiliary
                 ;register.
```



Example 3. Indirect Addressing—Decrement by 1

```
ADD *-,8         ;Operates as in Example 1 , but in
                 ;addition, the current auxiliary register
                 ;is decremented by one.
```

Example 4. Indirect Addressing—Increment by Index Amount

```
ADD *0+,8        ;Operates as in Example 1 , but in
                 ;addition, the content of register AR0
                 ;is added to the current auxiliary
                 ;register.
```

Example 5. Indirect Addressing—Decrement by Index Amount

```
ADD *0-,8        ;Operates as in Example 1 , but in
                 ;addition, the content of register AR0
                 ;is subtracted from the current auxiliary
                 ;register.
```

4.6 Assembly Programming Using the C2xx DSP Instruction Set

The complete detailed instruction set for the C2xx DSP core can be found in the Texas Instruments TMS320F/C24x DSP Controllers Reference Guide: CPU and Instruction Set; Literature Number: SPRU160C. This reference guide contains a complete descriptive listing on syntax, operands, binary opcode, instruction execution order, status bits affected by the instruction, number of memory words required to store the instruction, and clock-cycles used by the instruction. The Texas Instruments documentation on the assembly instruction set is very well written. Each assembly instruction has a complete explanation of the instruction, all optional operands, and several examples of the instructions used. Since including the instruction set and complete documentation would make this book excessively long, we will assume the reader has access to the documentation referred to above.

We will therefore focus on developing code, not the instruction set itself. Each command starts with the basic assembly instruction. Each command supports specific addressing modes and options. For example, the ADD command will work with direct, indirect, and immediate addressing. In addition to the basic command, many instructions have additional options that may be used with the instruction. For example, the ADD command supports left shifting of the data before it is added to the accumulator.

The following is the instruction syntax for the ADD command:

```
ADD dma [, shift]          ; Direct addressing
ADD dma, 16                ; Direct with left shift of 16
ADD ind [, shift [, ARn]]  ; Indirect addressing
ADD ind, 16 [, ARn]        ; Indirect with left shift of 16
ADD #k                     ; short immediate addressing
ADD #lk [, shift]          ; Long immediate addressing
```

The following is a list of the various notations used in C2xx syntax examples:

Italics Italic symbols in instruction syntax represent variables.

Example:

LACC dma, you can use several ways to address the dma (data memory address).

```
LACC *
```

Or

```
LACC 200h
```

Or

LACC v *; where "v" is any variable assigned to data memory*
*Where *, 200h, and v are the data memory addresses*

Boldface Characters Boldface characters must be included in the syntax.

Example:

LAR dma, 16 *; direct addressing with left shift of 16*

LAR AR1, 60h, 16 *; load auxiliary AR1 register with the memory contents of 60h that was left shifted 16 bits*

Example:

LACC dma, [shift] *; optional left shift from 0, 15; defaults to 0*

LACC main_counter, 8 *; shifts contents of the variable "main_counter" data 8 places to the left before loading accumulator*

[ ] An optional operand may be placed in the placed here.

Example:

LACC ind [, shift [, AR n]_] *Indirect addressing*

*LACC **
; load Accum. W/contents of the memory
; Location pointed to by the current AR.

*LACC *, 5*
; load Accum. With the contents of the memory
; Location pointed to by the current AR after
; The memory contents are left shifted by 5
; Bits.

*LACC *, 0, AR3*
; load Accum. With the contents of the memory
; Location pointed to by the current AR after
; The memory contents are left shifted by 5
; Bits. Now you have the option of choosing
; A new AR. In this case, AR3 will become the
; New AR.

[, x1 [, x2]] Operands x1 and x2 are optional, but you cannot include x2 without also including x1.

It is optional when using indirect addressing to modify the data. Once you supply a left shift value from 0…15 (even a shift of 0), then you have the option of changing to a new current auxiliary register (AR).

The # sign is prefix that signifies that the number used is a constant as opposed to memory location.

Example:

```
RPT #15          ; this syntax is using short immediate addressing. It will repeat the next
                 instruction 15+1 times.
LACC #60h        ; this will load the accumulator with the
                 ; Constant 60h
LACC 60h         ; However, this instruction will load the
                 ; Accumulator with the contents in the data
                 ; Memory location 60h, not the constant #60h
```

We will now provide a few examples of using the instruction set. Example 2.1 performs a few arithmetic functions with the DSP core and illustrates the nature of assembly programming. Programming with the assembly instruction set is somewhat different than languages such as C. In a high-level language, to add two numbers we might just code "c = a + b". In assembly, the user must be sure to code everything that needs to happen in order for a task to be executed. Take the following example:

Example 2.1 - Add the two numbers "2" and "3":

```
LDP #6h          ; loads the proper DP for dma 300h
SPLK #2, 300h    ; store the number "2" in memory address 300h
LACL #3          ; load the accumulator with the number "3"
ADD 300h         ; adds contents of 300h ("2") to the contents
                 ; of the accumulator ("3"); accumulator = 5
Another way:
LDP #6h          ; loads the proper DP for dma 300h
SPLK #2h, 300h   ; store the number "2h" in memory address
                 ; 300h
SPLK #3h, 301h   ; stores the number "3h" into memory address
                 ; 301h
```

```
LACL 300h          ; load the accumulator with the contents in
                   ; Memory location 300h
ADD 301h           ; adds contents of memory address 301h ("3h")
                   ; To the contents of the accumulator ("2h")
                   ;accumulator = 5h
```

Looping algorithms are very common in all programming languages. In high-level languages, the "For" and "While" loops can be used. However, in assembly, we need a slightly different approach to perform a repeating algorithm. The following example is an algorithm that stores the value "1" to memory locations 300h, 301h, 302h, 303h, and 304h.

Example 2.2- Looping Algorithm Using the Auxiliary Register

```
LAR AR0, #4        ; load auxiliary register 0 with #4
LAR AR1, #300h     ; this AR will be used as a memory pointer
LACL #1h           ; loads "1" into the accumulator
LOOPER MAR *, AR1  ; makes AR1 the next current AR
SACL *+, AR0       ; writes contents of accumulator to address
                   ; pointed to by AR1, the "+" increments AR1
                   ; By 1, next current AR is AR0
BANZ LOOPER        ; branch to LOOPER while current AR is not 0;
                   decrements current AR by 1 and branches
                   ; back to LOOPER
```

One might wonder if assembly language is so tedious to use, why not just program in a high-level language all the time. When code written in a high level language is compiled into assembly, the length of the code increases substantially. For example, if an assembly program takes up 50 lines, the same program written in C might take 150 lines after it is compiled. For this reason, code written in assembly almost always executed faster and uses less memory than high-level language code.

CONTENTS

UNIT V

FPGA

5.1 Introduction to Field Programmable Gate Arrays – CPLD Vs FPGA – Types of FPGA – Xilinx, XC3000 series - Configurable logic Blocks (CLB) – Input / Output Block (IOB) – Programmable Interconnect Point (PIP) – Xilinx 4000 series – HDL programming –overview of Spartan 3E and Virtex II pro FPGA boards- case study

Introduction to Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are digital ICs (Integrated Circuits) that enable the hardware design engineer to program a customized Digital Logic as per his/her requirements. The term "Field Programmable" implies that the Digital Logic of the IC is not fixed during its manufacturing (or fabrication) but rather it is programmed by the end-user (designer).

In order to provide this programmability, an FPGA consists of Configurable (or Programmable) Logic Blocks and configurable interconnects between these blocks. This configurable Logic and the Interconnections (Routing) of FPGAs makes them general purpose and flexible but at the same time, it also makes them slow and power hungry when compared to a similar calibre ASIC with Standard Cells.

It has been more than three decades since the introduction of FPGAs into the market and in this long span, they have undergone a severe technological advancement and gained a continuously growing popularity.

PLD (Programmable Logic Device)

Before diving into the main topic, I want to briefly discuss the concept of Programmable Logic Devices. So, what is a PLD. It is an IC containing a large number of Logic gates and Flip-flops that can be configured by the user to implement a wide variety of functions.

The simplest of Programmable Logic Devices consists of an array of AND & OR gates and the logic of these gates and their interconnections can be configured by a programming process.

PLDs are particularly useful when an engineer wants to implement a customized logic and is restricted by the pre-configured integrated circuits. PLDs provide a way to implement a custom digital circuit through the power of hardware configuration rather than implementing it using a software.

Different Types of PLDs

Basically, PLDs can be categorized into three types. They are:

- Simple Programmable Logic Devices (SPLD)
- Complex Programmable Logic Devices (CPLD)
- Field Programmable Gate Arrays (FPGA)

The Simple Programmable Logic Devices are further divided into:

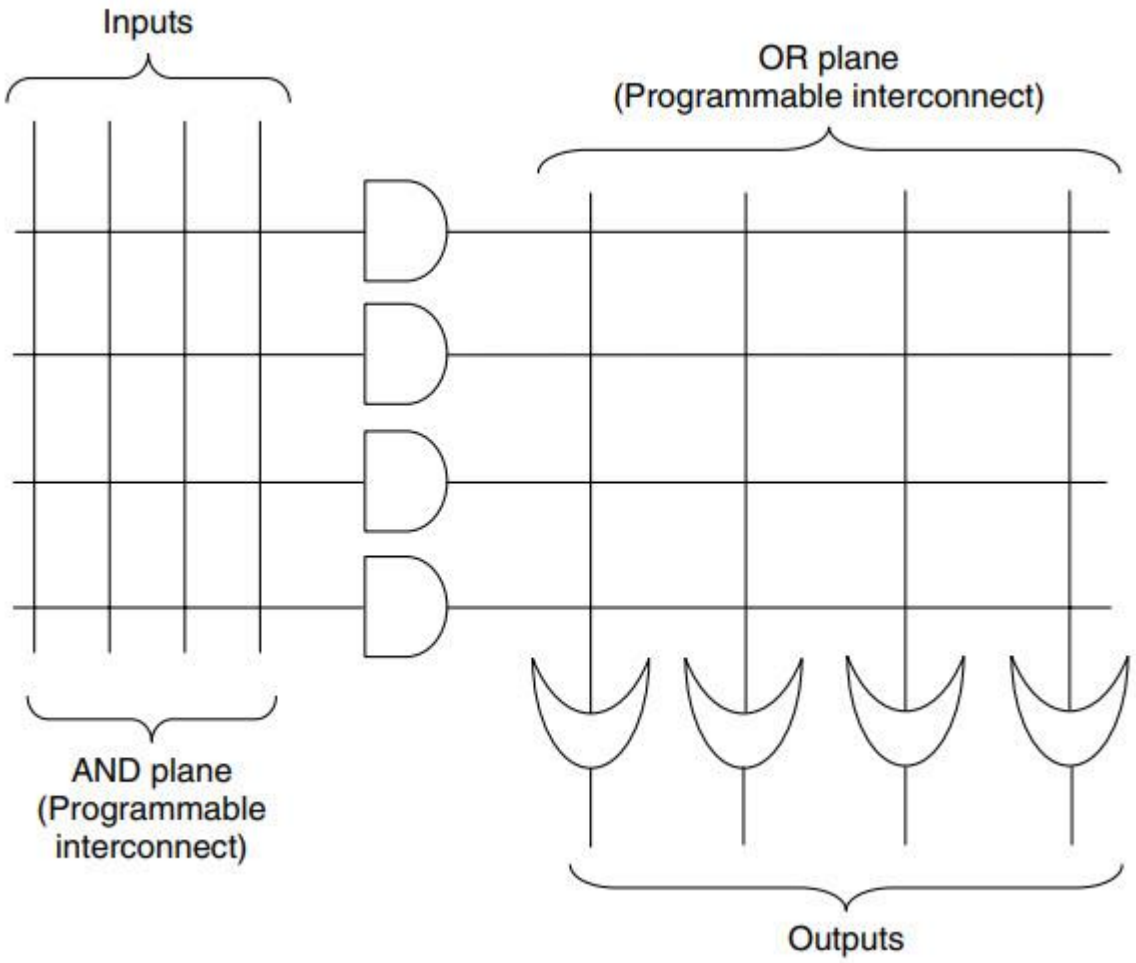
- Programmable Logic Array (PLA)
- Programmable Array Logic (PAL)
- Generic Array Logic (GAL)

Let us now see some basic details about all these PLDs.

Programmable Logic Array (PLA)

A PLA consists of an AND gate plane with programmable interconnects and an OR gate plane with programmable interconnects. The following is a simple four input – four output PLA with AND & OR gates.

Any input can be connected to any AND gate by connecting the horizontal and vertical interconnect lines. The outputs from different AND gates can then be applied to any of the OR gates with programmable interconnects.



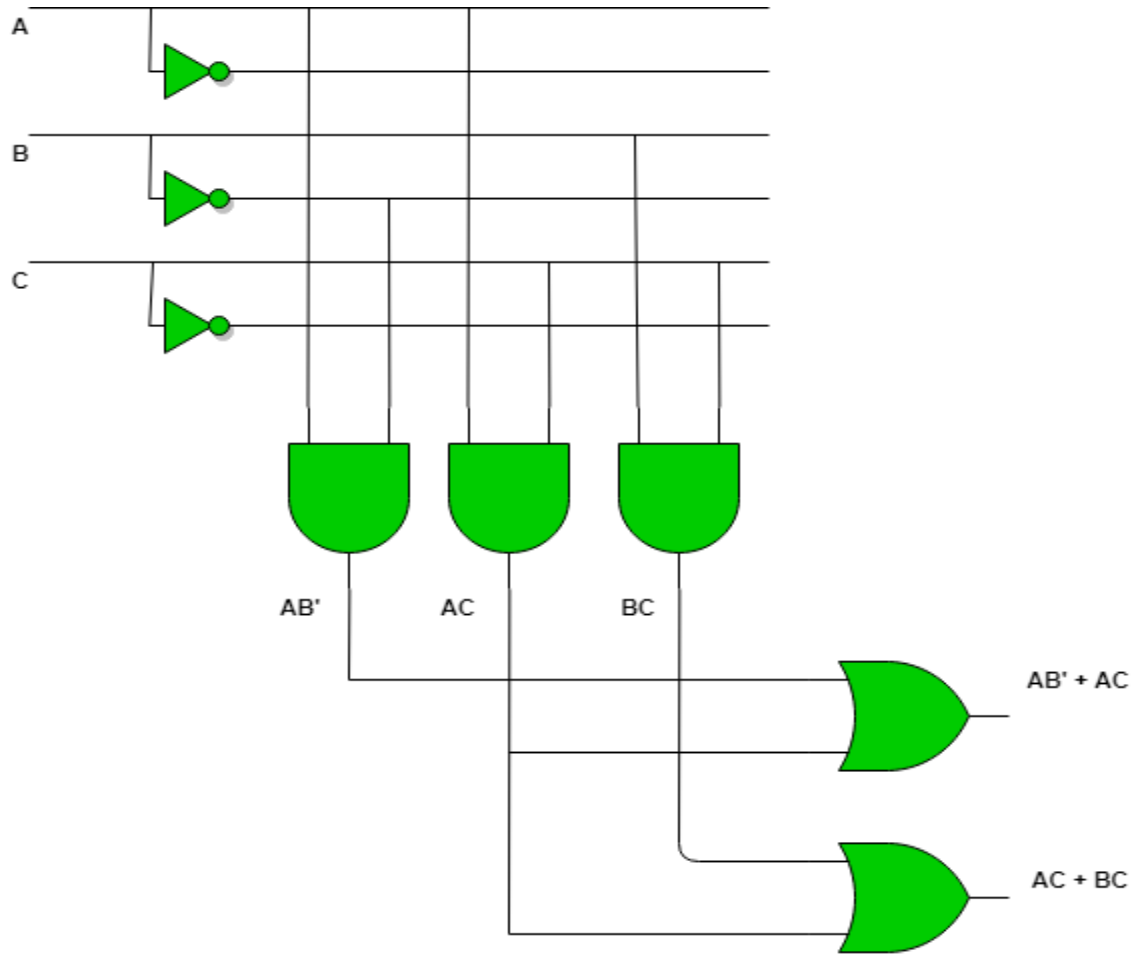


Fig PLA circuits

Programmable Array Logic (PAL)

A PAL is similar to the PLA but the difference is that in PAL, only the AND gate plane is programmable while the OR gate plane is fixed during fabrication. Even though PALs are less flexible than PLAs, they eliminate the time delays associated with programmable OR Gates.

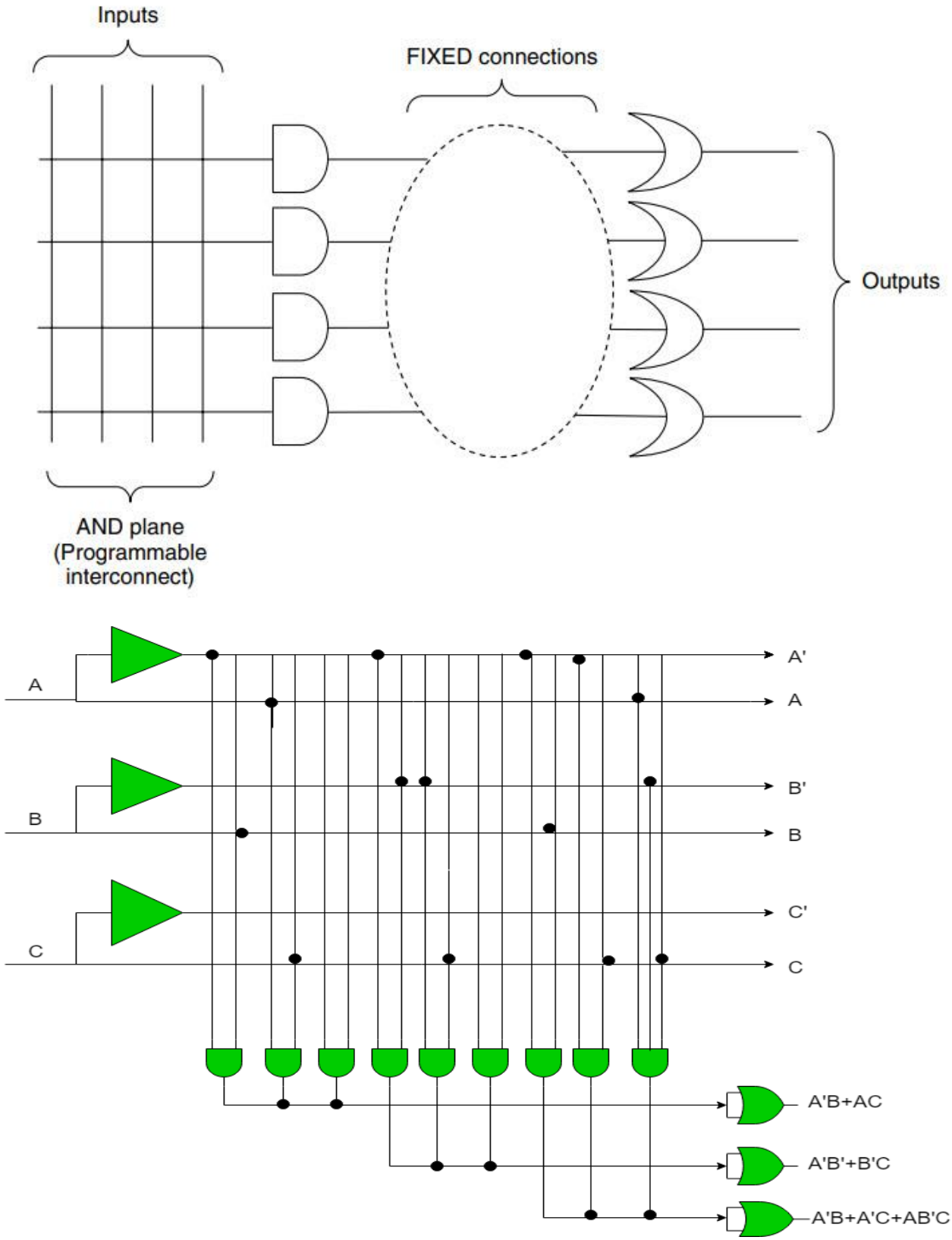


Fig. PAL Circuits

Generic Array Logic (GAL)

Architecture wise, a GAL is similar to a PAL but the difference lies in programmable structure. PALs use PROM, which is one-time programmable, while GAL uses EEPROM, which can be reprogrammed.

5.2 CPLD Vs FPGA

CPLD is often used for simple logic applications. It contains only a few blocks of logic and reaches up to 100. Having said that, CPLDs are considered as 'coarse-grain' type of devices. CPLDs are cheap and it also offers a much faster input to output duration because of its simpler, 'coarse grain' architecture.

FPGAs are cheaper per gate but expensive when it comes to package.

Working with FPGAs requires special procedures as it is RAM based. To program the device, you have to first describe the 'logic function' with the use of computer, either by drawing a schematic or simply describing the function on a text file.

Compilation of the 'logic function' usually requires a software. It creates a binary file to be downloaded into the FPGA and then the chip will behave just what you have instructed in the 'logic function'.

1. FPGA contains up to 100,000 of tiny logic blocks while CPLD contains only a few blocks of logic that reaches up to a few thousands.

2. In terms of architecture, FPGAs are considered as 'fine-grain' devices while CPLDs are 'coarse-grain'.

3. FPGAs are great for more complex applications while CPLDs are better for simpler ones.

4. FPGAs are made up of tiny logic blocks while CPLDs are made of larger blocks.

5. FPGA is a RAM-based digital logic chip while CPLD is EEPROM-based.

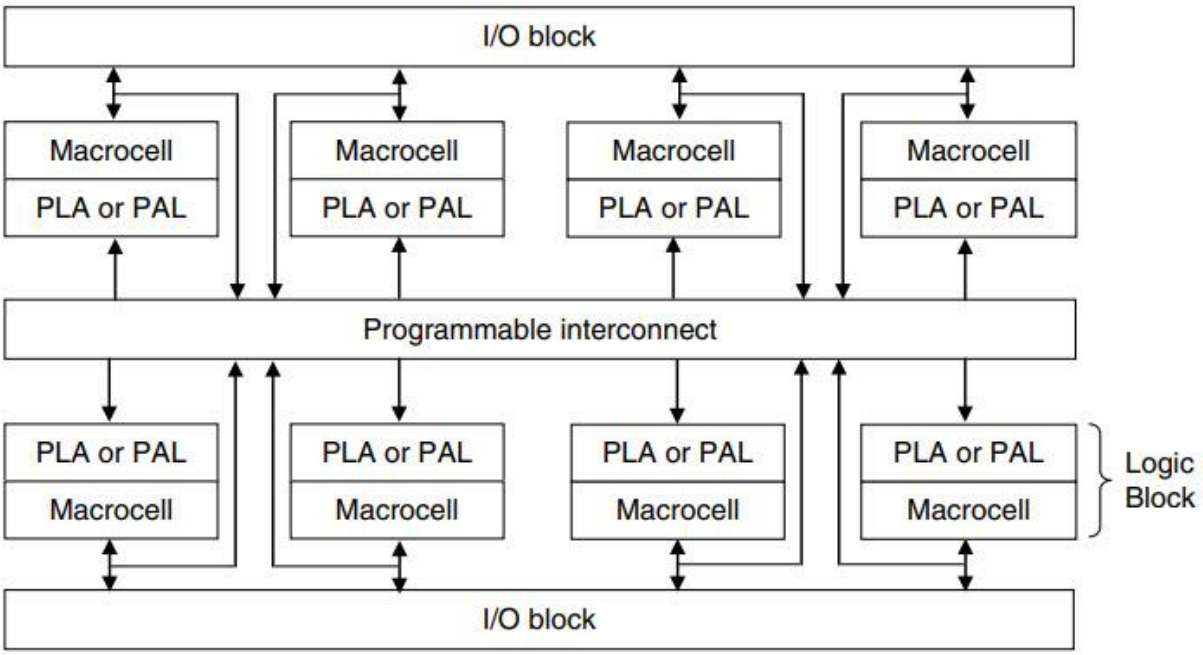
6. Normally, FPGAs are more expensive while CPLDs are much cheaper.

7. Delays are much more predictable in CPLDs than in FPGAs.

Complex Programmable Logic Devices (CPLD)

Moving up from SPLD devices, we get CPLD. It is developed on top of SPLD devices to create mush larger and complex designs. A CPLD consists of a number logic blocks (or functional blocks), which internally consists of either a Pal or a PAL along with a Macrocell.

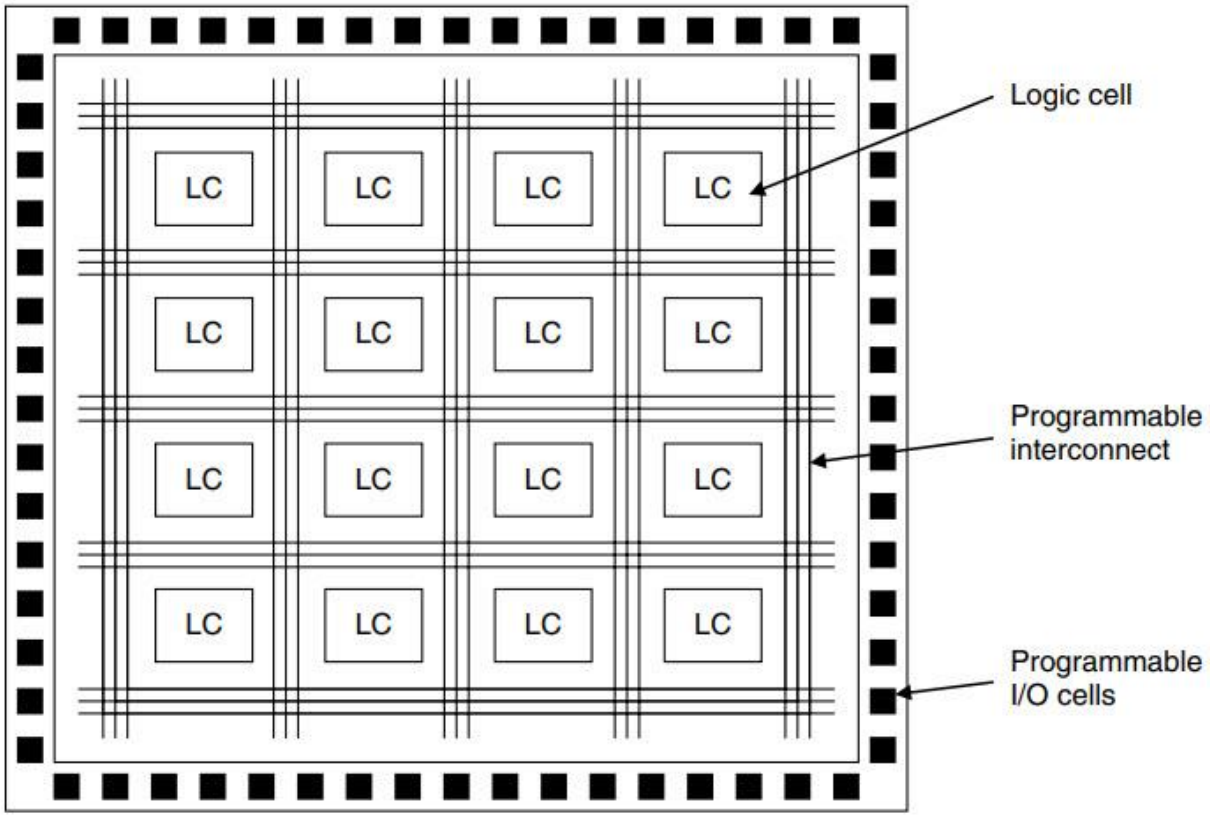
Macrocell consists of any additional circuitry and signal polarity control to provide true signal or its complement



Field Programmable Gate Arrays (FPGA)

Complexity wise, CPLD are much more complex than SPLDs. But FPGA are even more complex than CPLDs. The architecture of an FPGA is completely different as it consists of programmable Logic Cells, programmable interconnects and programmable IO blocks.

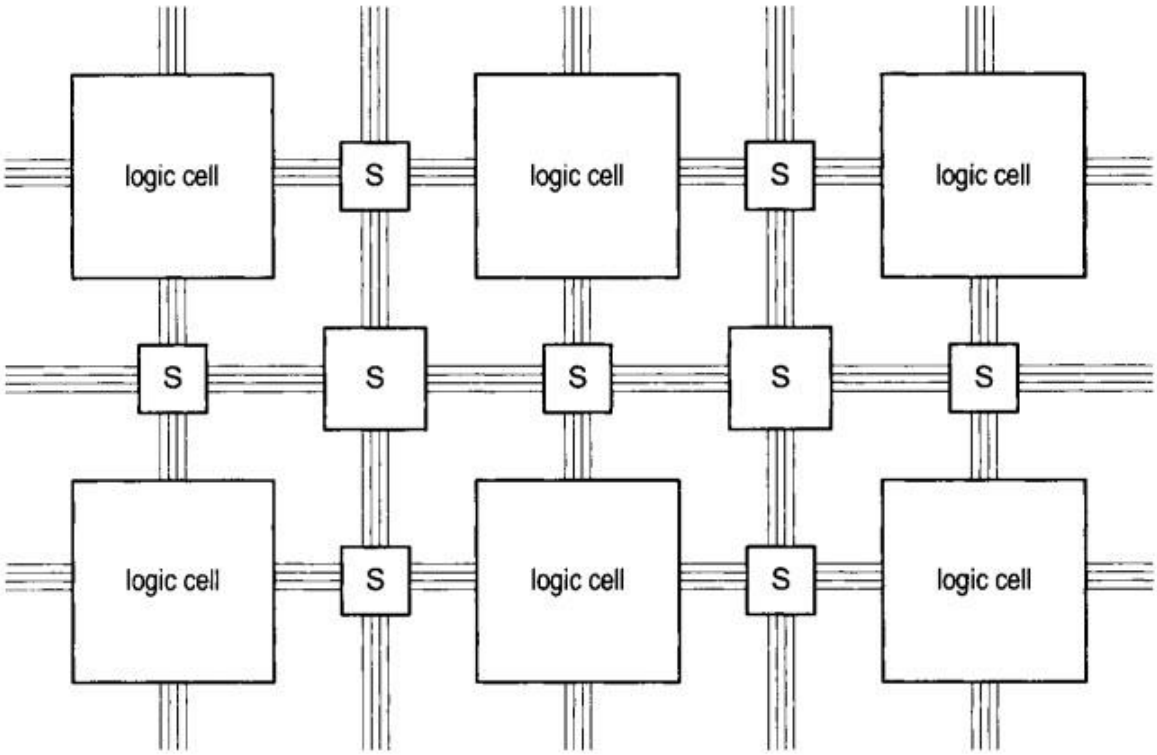
Field Programmable Gate Arrays or FPGAs in short are pre-fabricated Silicon devices that consists of a matrix of reconfigurable logic circuitry and programmable interconnects arranged in a two-dimensional array. The programmable Logic Cells can be configured to perform any digital function and the programmable interconnects (or switches) provide the connections among different logic cells.



Using an FPGA, you can implement any custom design by specifying the logic or function of each logic block and setting the connection of each programmable switch. Since this process of designing a custom circuit is done in the field rather than in a fab, the device is known as “Field Programmable”.

The following image shows a typical internal structure of an FPGA in a very broad sense.

As you can see, the core of the FPGA is made up of configurable logic cells and programmable interconnections. These are surrounded by a number of programmable IO blocks, which are used to talk to the external world.



Components of an FPGA

Let us now take a closer look at the structure of an FPGA. Typically, an FPGA consists of three basic components. They are:

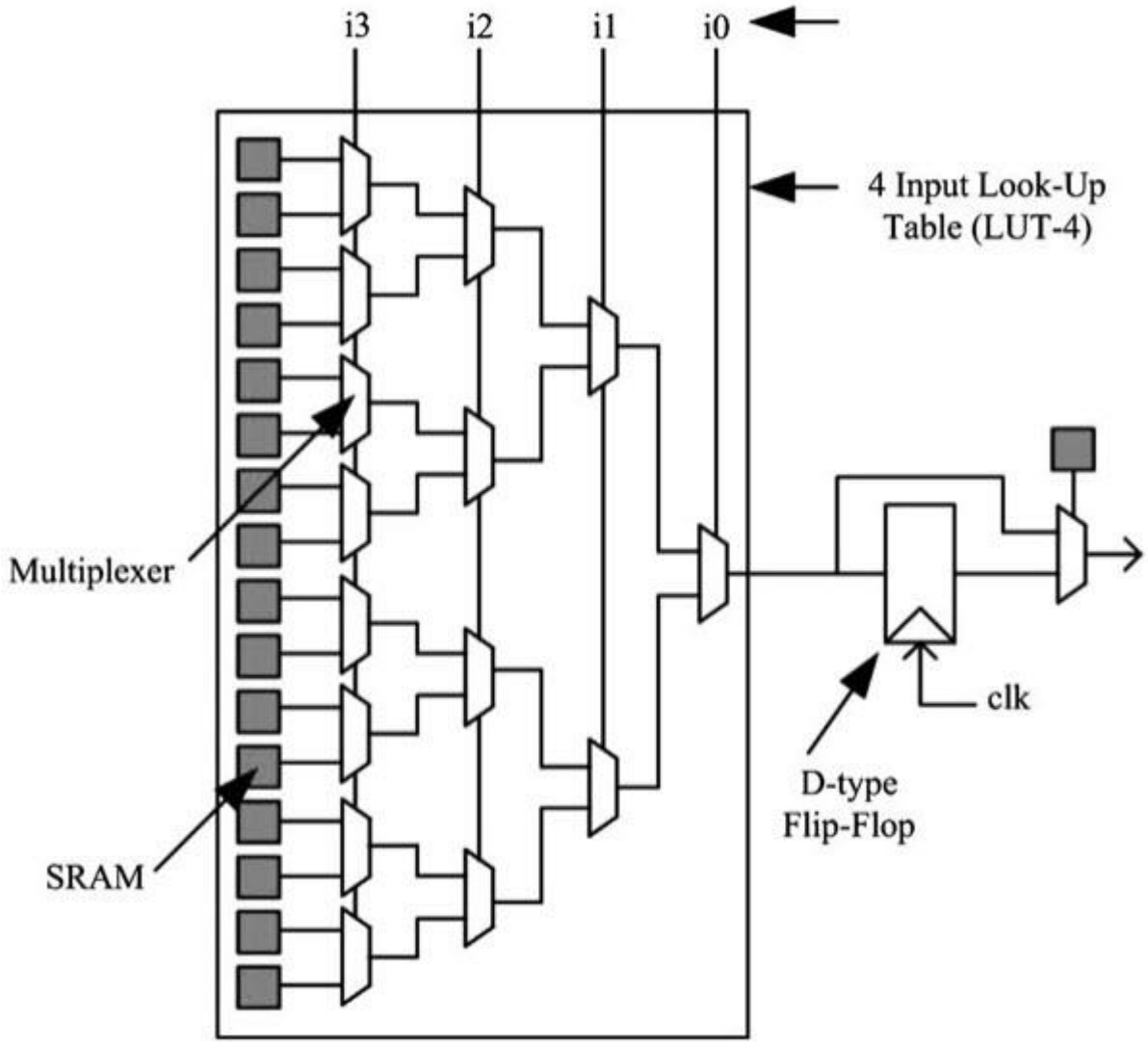
- Programmable Logic Cells (or Logic Blocks) – responsible for implementing the core logic functions.
- Programmable Routing – responsible for connecting the Logic Blocks.
- IO Blocks – which are connected to the Logic Blocks through the routing and help to make external connections

Logic Block

The Logic Block in Xilinx based FPGAs are called as Configurable Logic Blocks or CLB while the similar structures in Altera based FPGAs are called Logic Array Blocks or LAB. Let us use the term CLB for this discussion. A CLB is the basic component of an FPGA, which provides both the logic and storage functionalities. The basic logic block can be anything like a transistor,

a NAND gate, Multiplexors, Look-up Table (LUT), a PAL like structure or even a processor. Both Xilinx and Altera use Look-up Table (LUT) based logic blocks to implement the logic as well as the storage functionalities.

A Logic Block can be made up of a single Basic Logic Element or a set of interconnected Basic Logic Elements, where a Basic Logic Element is a combination of a Look-up table (which is in turn made up of SRAM and Multiplexors) and a Flip-flop



A LUT with 'n' inputs consists of 2^n configuration bits, which are implemented by SRAM Cells. Using these 2^n SRAM Bits, the LUT can be configured to implement any logical function.

Routing

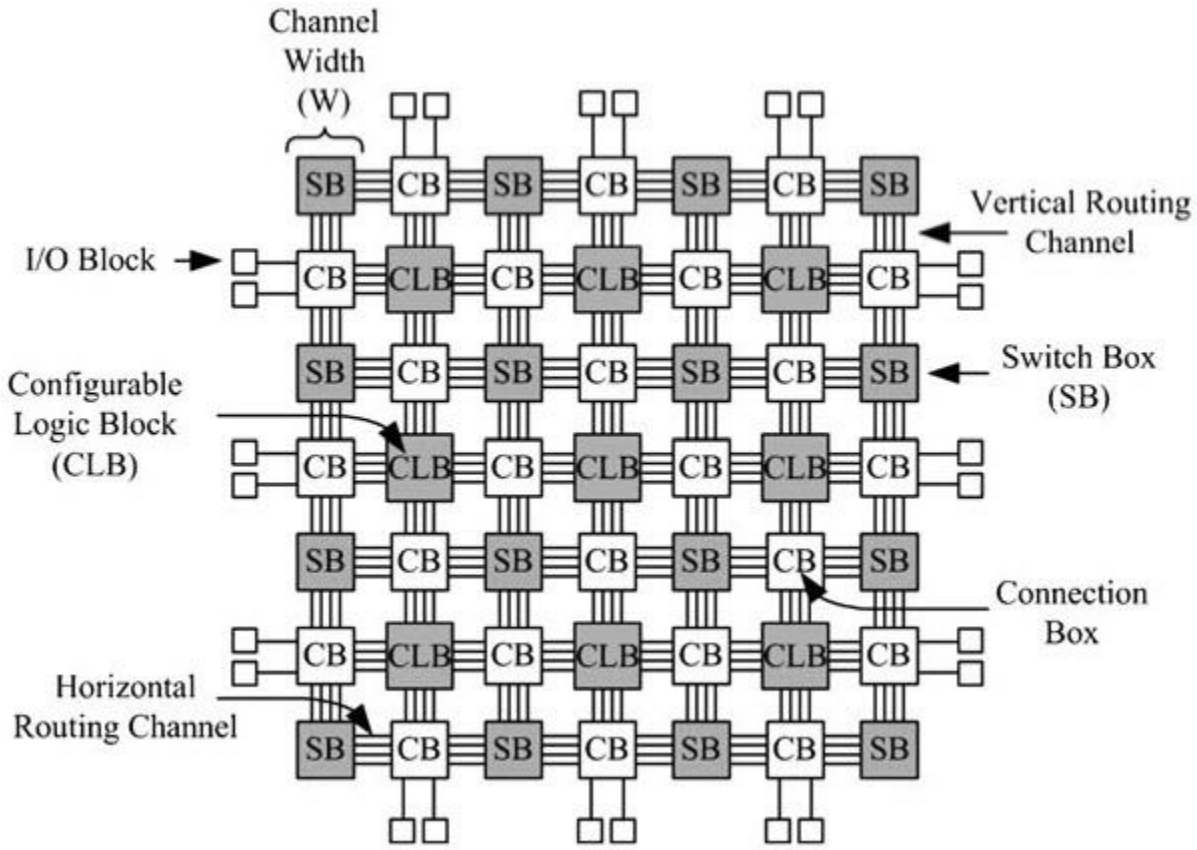
If the computational functionality is provided by the Logic Blocks, then the programmable routing network is responsible for interconnection these logic blocks. The Routing Network provides interconnections between one logic block to other as well as between the logic block and the IO Block to completely implement a custom circuit.

Basically, the routing network consists of connecting wires with programmable switches, which can be configured using any of the programming technologies. There are basically two types of routing architectures. They are:

- Island Style Routing (also known as Mesh Routing)
- Hierarchical Routing

In island style routing architecture, the logic blocks are arranged in a two-dimensional array and are interconnected using a programmable routing network. This type of routing is widely used in commercial FPGAs.

Many logic blocks are confined to a local set of connections and hierarchical routing architecture makes use of this feature by dividing the logic blocks into several groups or clusters. If the logic blocks are residing in the same cluster, then the hierarchical routing connects them in a low level of hierarchy.



5.3 Types of FPGA

FPGA Programming Technologies

We have talked about the reprogrammable architecture of FPGAs quite a bit but now let us see some of the most commonly used programming techniques that is responsible for such reconfigurable architecture.

The following are three of the well-known programming technologies used in FPGAs.

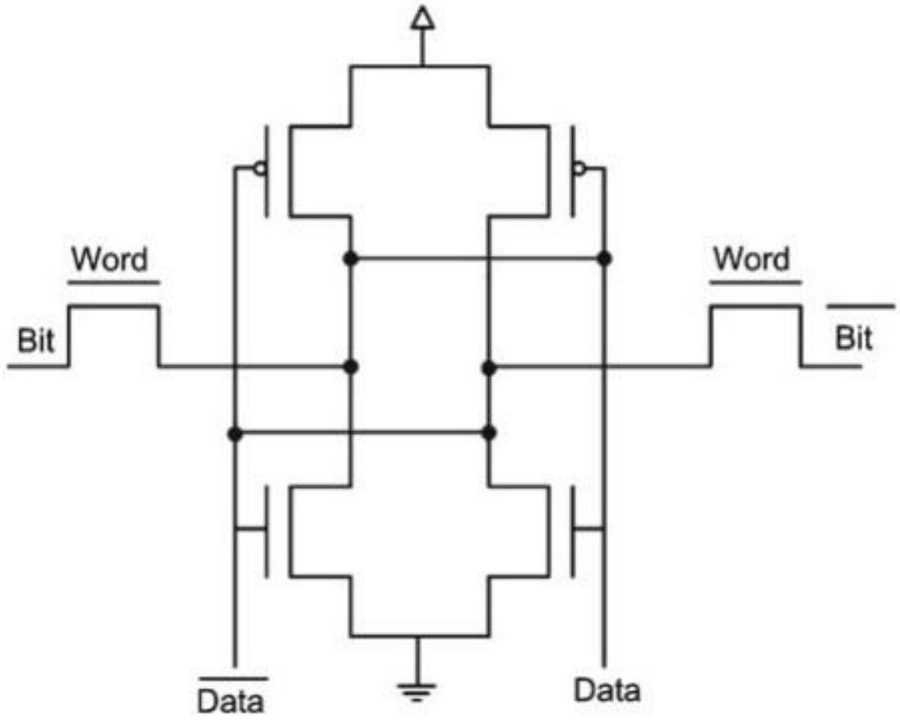
- SRAM
- EEPROM / Flash
- Anti-Fuse

Other technologies include EPROM and Fusible Link but they are used in CPLDs and other PLDs but not in FPGAs, Hence, let us keep the discussion limited to FPGA related programming technologies.

SRAM

We know that there are two types of semiconductor RAM called the SRAM and DRAM. SRAM is short for Static RAM while DRAM is short for Dynamic Ram. SRAM is designed using transistors and the term static means that the value loaded on a basic SRAM Memory Cell will remain the same until deliberately changed or when the power is removed.

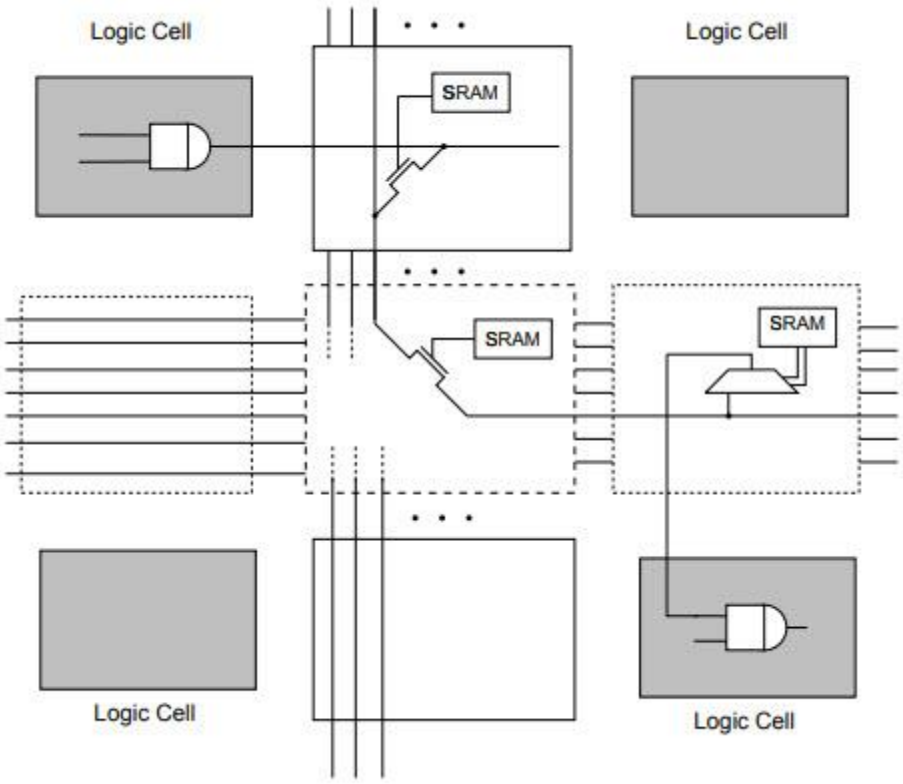
A typical 6 transistor SRAM Cell to store 1 bit is shown in the following image.



This is in contrast to the DRAM, which consists of a combination of a transistor and a capacitor. The term Dynamic refers to the fact that the value loaded in the basic DRAM Memory Cell is

valid until there is charge in the capacitor. As capacitor loses its charge over time, the memory cell has to be periodically recharged to maintain the charge. This is also known as refreshing.

Many FPGA vendors implement Static Memory Cells in SRAM based FPGAs for programming. SRAM based FPGAs are used to program both the logic cells and the interconnects and they have become quite predominant due to their re-programmability and use of CMOS technology, which is known for its low dynamic power consumption, high speed and tighter integration.



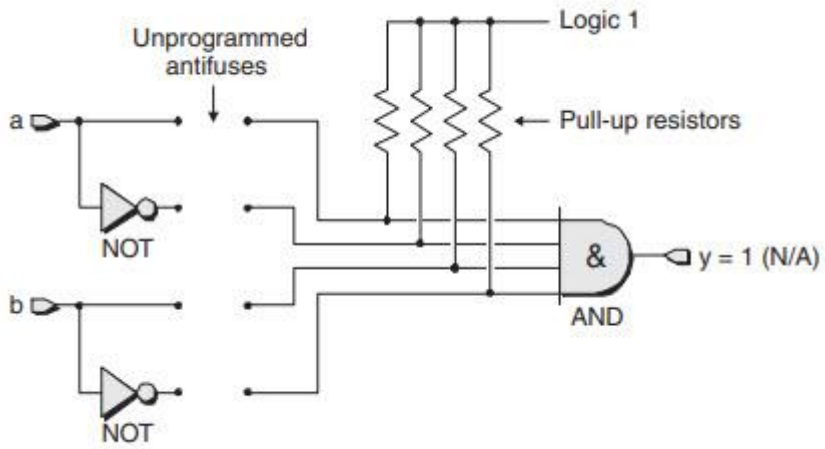
EEPROM / Flash

A close alternative to SRAM based programming technology is based on EEPROM or Flash programming technologies. The main advantage of flash-based programming is its non-volatile nature. Even though flash supports re-programmability, the number of times this can be done is very small when compared to an SRAM technology.

Anti-Fuse

The anti-fuse programming technology is an old technique of producing one-time programmable devices. They are implemented using a link called the antifuse, which in its unprogrammed state has a very high resistance and can be considered an open circuit.

When programming, a high voltage and current is supplied to the input. As a result, the antifuse, which is initially in the form of amorphous silicon (basically an insulator with very high resistance) linking two metal tracks, comes to life by converting to a conducting polysilicon.



When compared to the other two technologies, the antifuse one occupies the least amount of space but comes only as one-time programmable option.

Property	OTP FPGA	MTP FPGA
Speed	smaller	larger
Power Consumption	lower	higher
Working Environment (Radiation)	Radiation hardened	NO radiation hardened
Design Cycle	Programmed once only	Many times
Price	Almost the same	Almost the same
Reliability	More (single Chip)	Less (2 Chips, FPGA & PROM)
Security	More secure	Less secure

5.4 Xilinx

Xilinx is the inventor of the FPGA, programmable SoCs, and now, the ACAP. Xilinx delivers the most dynamic processing technology in the industry.

Xilinx, Inc. (/ˈzɪlɪŋks/ *ZEE-links*) was an American technology and semiconductor company that primarily supplied programmable logic devices. The company was known for inventing the first commercially viable field-programmable gate array (FPGA) and creating the first fabless manufacturing model,

Xilinx was co-founded by Ross Freeman, Bernard Vonderschmitt, and James V Barnett II in 1984 and the company went public on the NASDAQ in 1989. AMD announced its acquisition of Xilinx in October 2020 and the deal was completed on February 14, 2022 through an all-stock transaction worth an estimated $50 billion.

Before 2010, Xilinx offered two main FPGA families: the high-performance Virtex series and the high-volume Spartan series, with a cheaper EasyPath option for ramping to volume production. The company also provides two CPLD lines: the CoolRunner and the 9500 series. Each model series has been released in multiple generations since its launch. With the introduction of its 28 nm FPGAs in June 2010, Xilinx replaced the high-volume Spartan family with the Kintex family and the low-cost Artix family.

5.5 XC3000 series

Features

Complete line of four related Field Programmable Gate Array product families - XC3000A, XC3000L, XC3100A, XC3100L

• Ideal for a wide range of custom VLSI design tasks - Replaces TTL, MSI, and other PLD logic - Integrates complete sub-systems into a single package - Avoids the NRE, time delay, and risk of conventional masked gate arrays

• High-performance CMOS static memory technology - Guaranteed toggle rates of 70 to 370 MHz, logic delays from 7 to 1.5 ns - System clock speeds over 85 MHz - Low quiescent and active power consumption

• Flexible FPGA architecture - Compatible arrays ranging from 1,000 to 7,500 gate complexity - Extensive register, combinatorial, and I/O capabilities - High fan-out signal distribution, low-skew clock nets - Internal 3-state bus capabilities - TTL or CMOS input thresholds - On-chip crystal oscillator amplifier

• Unlimited reprogrammability - Easy design iteration - In-system logic changes

• Extensive packaging options - Over 20 different packages - Plastic and ceramic surface-mount and pin-gridarray packages - Thin and Very Thin Quad Flat Pack (TQFP and VQFP) options

• Ready for volume production - Standard, off-the-shelf product availability - 100% factory pre-tested devices - Excellent reliability record

Complete Development System - Schematic capture, automatic place and route - Logic and timing simulation - Interactive design editor for design optimization - Timing calculator - Interfaces to popular design environments like Viewlogic, Cadence, Mentor Graphics, and others

5.6 Configurable logic Blocks

A configurable logic block (CLB) is the basic repeating logic resource on an FPGA. When linked together by routing resources, the components in CLBs execute complex logic functions, implement memory functions, and synchronize code on the FPGA.

CLBs contain smaller components, including flip-flops, look-up tables (LUTs), and multiplexers.

- **Flip-Flop**—A circuit capable of two stable states that represents a single bit. A flip-flop is the smallest storage resource on the FPGA. Each flip-flop in a CLB is a binary register used to save logic states between clock cycles on an FPGA circuit.
- **Look-up Table (LUT)**—A collection of gates hardwired on the FPGA. An LUT stores a predefined list of outputs for every combination of inputs. LUTs provide a fast way to retrieve the output of a logic operation because possible results are stored and then referenced rather than calculated. The LUTs in a CLB can also implement FIFOs and memory items in LabVIEW.
- **Multiplexer**—A circuit that selects between two or more inputs and then returns the selected input.

When you compile code to run on an FPGA target, LabVIEW implements much of the code using flip-flops, LUTs, and multiplexers.

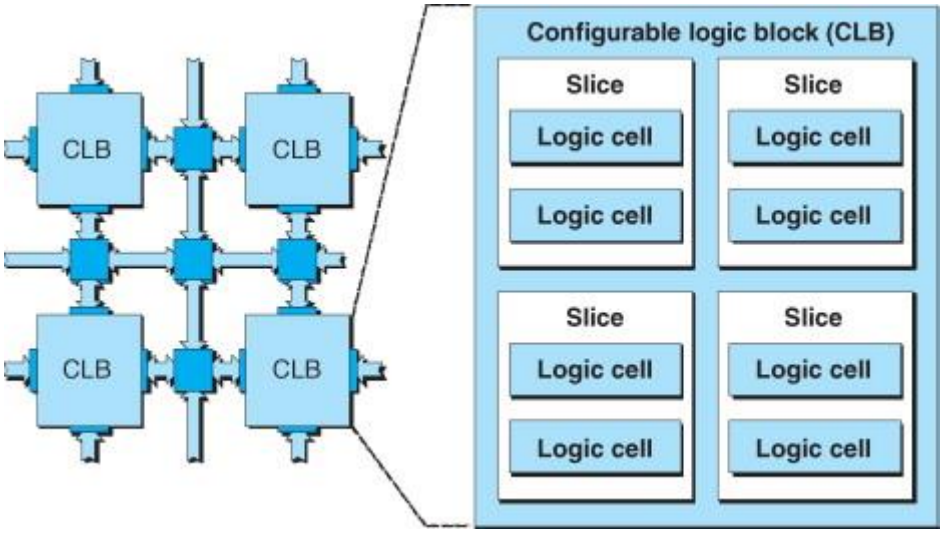


Figure Configurable logic Blocks

5.7 Input / Output Block (IOB)

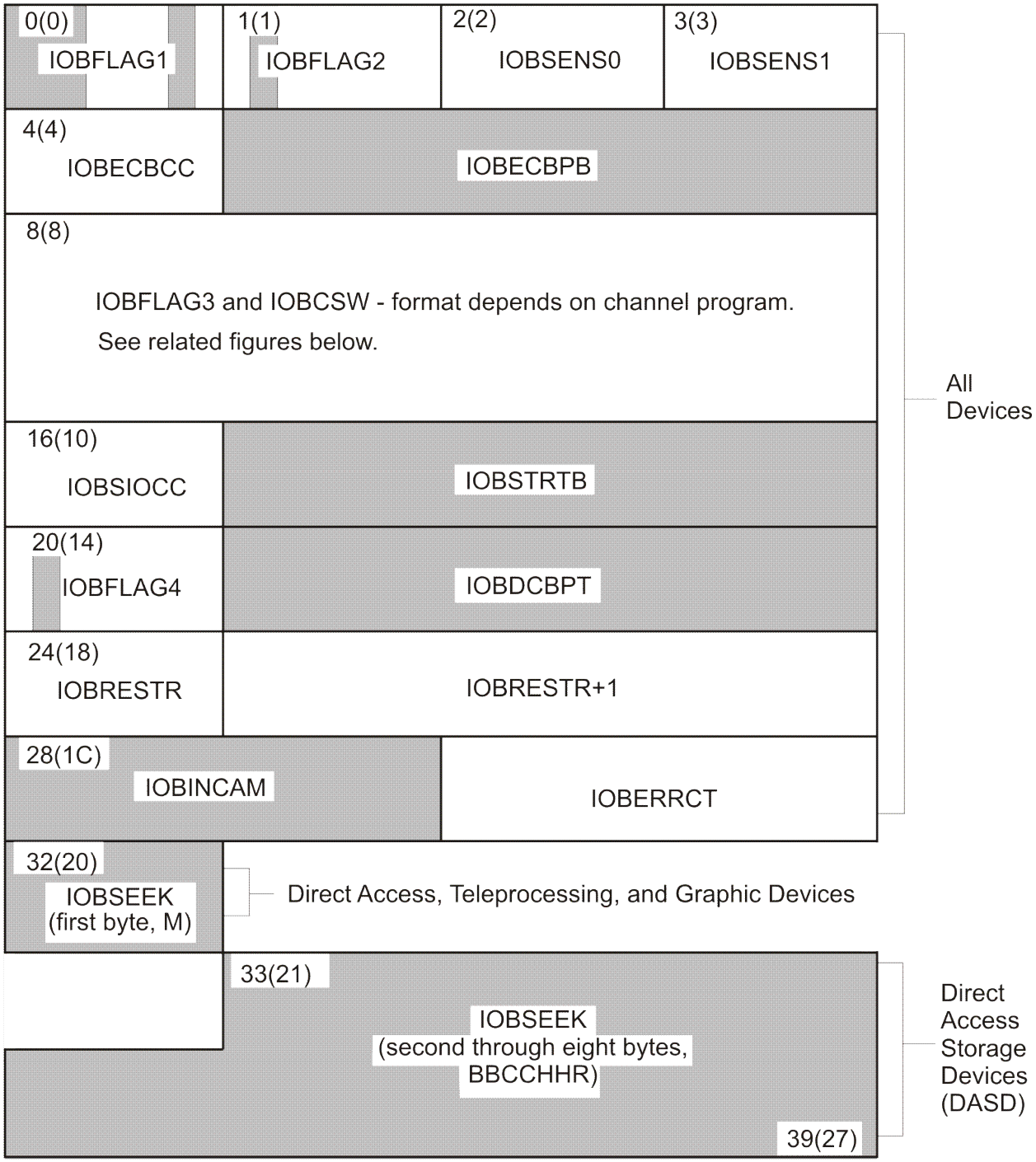
The input/output block (IOB) is used for communication between the problem program and the system. It provides the addresses of other control blocks, and maintains information about the channel program, such as the type of chaining and the progress of I/O operations. You must define the IOB and specify its address as the only parameter of the EXCP or EXCPVR macro instruction.

The input/output block (IOB) is not automatically constructed by a macro instruction; it must be defined as a series of constants and be on a word boundary. For unit-record and tape devices, the IOB is 32 bytes long. For direct access, teleprocessing, and graphic devices, 8 additional bytes must be provided. Use the system mapping macro IEZIOB, which expands into a DSECT, to help in constructing an IOB. IEZIOB fields that are not described here are not part of the programming interface.

In Figure 1 the shaded areas indicate fields in which you must specify information. The other fields are used by the system and must be defined as all zeros. You cannot place information into these fields, but you can examine them.

You do not have to set the following IOB fields to any particular value before issuing EXCP because the system itself sets them:

- IOBSENS0
- IOBSENS1
- IOBECBCC
- IOBCSW
- IOBSIOCC
- IOBCMD31



. Figure Input/Output Block (IOB) Format

IOBFLAG1 (1 byte)

Set bit positions 0, 1, 6, and 7. One-bits in positions 0 and 1 (IOBDATCH and IOBCMDCH) indicate data chaining and command chaining, respectively. (If you

specify both data chaining and command chaining, the system does not use error recovery routines except for the direct access and tape devices.) If an I/O error occurs while your channel program executes, a failure to set the chaining bits in the IOB that correspond to those in the CCW might make successful error recovery impossible. The integrity of your data could be compromised.

A one-bit in position 6 (IOBUNREL) indicates that the channel program is not a related request; that is, the channel program is not related to any other channel program. See bits 2 and 3 of IOBFLAG2 below.

If you intend to issue an EXCP or XDAP macro with a BSAM, QSAM, or BPAM DCB, you should turn on bit 7 (IOBSPSVC) to prevent access-method appendages from processing the I/O request.

IOBFLAG2 (1 byte)

If you set bit 6 in the IOBFLAG1 field to zero, bits 2 and 3 (IOBRRT3 and IOBRRT2) in this field must then be set to one of the following:

- 00, if any channel program or appendage associated with a related request might modify this IOB or channel program.
- 01, if the conditions requiring a 00 setting do not apply, but the CHE or ABE appendage might retry this channel program if it completes normally or with the unit-exception or wrong-length-record bits on in the CSW.
- 10 in all other cases.

The combinations of bits 2 and 3 represent related requests,known as type 1 (00), type 2 (01), and type 3 (10). The type you use determines how much the system can overlap the processing of related requests. Type 3 allows the greatest overlap, normally making it possible to quickly reuse a device after a channel-end interruption. (Related requests that were executed on a pre-MVS system are executed as type-1 requests if not modified.)

IOBSENS0 and IOBSENS1 (2 bytes)

are set by the system when a unit check occurs. These are the first two sense bytes. Occasionally, the system is unable to obtain any sense bytes because of unit checks when sense commands are issued. In this case, the system simulates sense bytes by moving X'10FE' to IOBSENS0 and IOBSENS1.

The first six of these 16 bits have these device-independent meanings:

1... .... Command reject
.1.. .... Intervention required
..1. .... Bus out check
...1 .... Equipment check
.... 1... Data check
.... .1.. Overrun

The last ten of these 16 bits have device-dependent meanings. See appropriate hardware documentation.

If you wish to retrieve more than two sense bytes, supply an IOBE and IEDB

OBECBCC (1 byte)

The first byte of the completion code for the channel program. The system places this code in the high-order byte of the event control block when the channel program is posted complete.

OBECBPT (3 bytes)

The address of the 4-byte event control block (ECB) you have provided.

IOBFLAG3 (1 byte) and IOBCSW (7 bytes)

The system stores status information in these eight bytes

IOBSIOCC (1 byte)

If the channel program uses format 0 CCWs, bits 2 and 3 contain the start subchannel (SSCH) condition code for the instruction the system issues to start the channel program.

If this is a format 1 CCW channel program or is a zHPF channel program, then field IOBSIOCC is redefined as field IOBSTART, which contains the four byte starting address of the channel program to be executed

IOBSTRTB (3 bytes)

If the channel program uses format 0 CCWs, the three byte starting address of the channel program to be executed.

IOBFLAG4 (1 byte)

Set bit 3 (IOBCEF) to indicate whether you are supplying an IOB common extension (IOBE). If this bit is 1, then register 0 contains the IOBE address

IOBDCBPT (3 bytes)

The address of the DCB of the data set to be read or written by the channel program.

Reserved (1 byte)

Used by the system.

IOBRESTR+1 (3 bytes)

If a related channel program is permanently in error, this field is used to chain together IOBs that represent dependent channel programs.

IOBINCAM (2 bytes)

For magnetic tape, the amount by which the system increments the block count (DCBBLKCT) field in the device-dependent portion of the DCB. You can alter these bytes at any time. For forward operations, these bytes should contain a binary positive integer (usually +1); for backward operations, they should contain a binary negative integer.

IOBERRCT (2 bytes)

Used by the system.

IOBSEEK (first byte, M)

For direct access devices, the extent entry in the data extent block that is associated with the channel program (0 indicates the first entry; 1 indicates the second, and so forth).

IOBSEEK (last 7 bytes, BBCCHHR)

For direct access devices, the seek address for your channel program.

5.8 Programmable Interconnect Point (PIP)

Programmable Interconnect Points (PIPs)

•Also known as Configurable Interconnect Points (CIPs)

Transmission gate connects to 2 wire segments–Controlled by configuration memory bit

•0 = wires disconnected

•1 = wires connected

PIPs

Break-point PIP–Connect or isolate 2 wire segments

•Cross-point PIP–Turn corners•Multiplexer PIP–Directional and buffered–Select 1-of-Ninputs for output

•Decoded MUX PIP –Nconfig bits select from 2Ninputs•Non-decoded MUX PIP – 1 config bit per input

•Compound cross-point PIP–Collection of 6 break-point PIPs

•Can route to two isolated signal nets.

5.9 Xilinx 4000 series

The XC4000E family of high-performance, high-density Field Programmable Gate Arrays (FPGAs) provides the benefits of custom CMOS VLSI, while avoiding the initial cost, time delay, and inherent risk of a conventional masked gate array. The XC4000E family combines architectural versatility, on-chip Select-RAM memory with edge-triggered and dual-port modes, increased speed, abundant routing resources, and new, sophisticated software to achieve fully automated imple-mentation. The FPGAs are customized by loading configuration data into the internal memory cells. The FPGA can either actively read its configuration data out of external serial or byte-parallel PROM (master mode), or the configuration data can be written into the FPGA (slave and peripheral mode). The XC4000E family can run at synchronous system clock rates of up to 70 MHz and internal performance in excess of 150 MHz.

Features

Third Generation Field-Programmable Gate Arrays

Select-RAM TM memory: on-chip ultra-fast RAM with - synchronous write option - dual-port RAM option

Fully PCI compliant

Abundant flip-flops

Flexible function generators

Dedicated high-speed carry-propagation circuit

Wide edge decoders (four per edge)

Hierarchy of interconnect lines

Internal 3-state bus capability

8 global low-skew clock or signal distribution network

Flexible Array Architecture

Programmable logic blocks and I/O blocks

Programmable interconnects and wide decoders

Sub-micron CMOS Process

High-speed logic and Interconnect

Low power consumption

Systems-Oriented Features

IEEE 1149.1-compatible boundary scan logic support

Programmable output slew rate (2 modes)

Programmable input pull-up or pull-down resistors

12-mA sink current per output

24-mA sink current per output pair

Configured by Loading Binary File

Unlimited reprogrammability

Six programming modes

Readback capability

Backward Compatible with XC4000 Family

XACTstep Development System runs on '386/'486/ Pentium-type PC, Sun-4, and Hewlett-Packard 700 series

Interfaces to popular design environments including VIEWlogic, Mentor Graphics and OrCAD

Fully automatic partitioning, placement and routing

Interactive design editor for design optimization

Unified Libraries, including 288 soft macros and 34 Relationally Placed Macros (RPMs)

RAM/ROM compiler

The XC4000E family is supported by powerful and sophisticated software, covering every aspect of design from schematic or behavioral entry, floor planning, simulation, automatic block placement and routing of interconnects, to the creation, downloading, and read back of the configura-tion bit stream.

The Xilinx XC4000E family includes three major configurable elements: configurable logic blocks (CLBs), input/output blocks, and interconnects.

The CLBs provide the functional elements for constructing user's logic. The IOBs provide the interface between the package pins and internal signal lines.

The programmable interconnect resources provide routing paths to connect the inputs and outputs of the CLBs and IOBs onto the appropriate networks.

Customized configuration is established by programming internal static memory cells that determine the logic functions and internal connections implemented in the FPGA.

Specifications	
Device	**XC4013E**
Aproximate Gate Count	13,000
CLB Matrix	24 x 24
Number of CLBs	567
Number of Flip-Flops	1,536
Max. Decode Inputs per Side	72
Max. RAM Bits	18,432
Number of IOBs	192
Horizontal Longlines	48
TBUFs per Longlines	26
PROM Size (bits)	247,96

XC4000E Series SRAM FPGA Overview

The XC4000E family of high-performance, high-density Field Programmable Gate Arrays (FPGAs) provides the benefits of custom CMOS VLSI, while avoiding the initial cost, time delay, and inherent risk of a conventional masked gate array. The XC4000E family combines architectural versatility, on-chip Select-RAM memory with edge-triggered and dual-port modes, increased speed, abundant routing resources, and new, sophisticated software to achieve fully automated imple-mentation. The FPGAs are customized by loading configuration data into the internal memory cells. The FPGA can either actively read its configuration data out of external serial or byte-parallel PROM (master mode), or the configuration data can be written into the FPGA (slave and peripheral mode). The XC4000E family can run at synchronous system clock rates of up to 70 MHz and internal performance in excess of 150 MHz.

Features

Third Generation Field-Programmable Gate Arrays

Select-RAM TM memory: on-chip ultra-fast RAM with - synchronous write option - dual-port RAM option

Fully PCI compliant

Abundant flip-flops

Flexible function generators

Dedicated high-speed carry-propagation circuit

Wide edge decoders (four per edge)

Hierarchy of interconnect lines

Internal 3-state bus capability

8 global low-skew clock or signal distribution network

Flexible Array Architecture

Programmable logic blocks and I/O blocks

Programmable interconnects and wide decoders

Sub-micron CMOS Process

High-speed logic and Interconnect

Low power consumption

Systems-Oriented Features

IEEE 1149.1-compatible boundary scan logic support

Programmable output slew rate (2 modes)

Programmable input pull-up or pull-down resistors

12-mA sink current per output
24-mA sink current per output pair

Configured by Loading Binary File

Unlimited reprogrammability
Six programming modes
Readback capability

Backward Compatible with XC4000 Family

XACTstep Development System runs on '386/'486/ Pentium-type PC, Sun-4, and Hewlett-Packard 700 series

Interfaces to popular design environments including VIEWlogic, Mentor Graphics and OrCAD
Fully automatic partitioning, placement and routing
Interactive design editor for design optimization
Unified Libraries, including 288 soft macros and 34 Relationally Placed Macros (RPMs)
RAM/ROM compiler

The XC4000E family is supported by powerful and sophisticated software, covering every aspect of design from schematic or behavioral entry, floorplanning, simulation, automatic block placement and routing of interconnects, to the creation, downloading, and readback of the

configura-tion bit stream. The Xilinx XC4000E family includes three major configurable elements: configurable logic blocks (CLBs), input/output blocks, and interconnects. The CLBs provide the functional elements for constructing user's logic. The IOBs provide the interface between the package pins and internal signal lines. The programmable interconnect resources provide routing paths to connect the inputs and outputs of the CLBs and IOBs onto the appropriate networks. Customized configuration is established by programming internal static memory cells that determine the logic functions and internal connections implemented in the FPGA.

Specifications	
Device	**XC4013E**
Aproximate Gate Count	13,000
CLB Matrix	24 x 24
Number of CLBs	567
Number of Flip-Flops	1,536
Max. Decode Inputs per Side	72
Max. RAM Bits	18,432
Number of IOBs	192
Horizontal Longlines	48
TBUFs per Longlines	26
PROM Size (bits)	247,960

Configurable Logic Blocks (CLB)

The principle CLB elements are shown in **Figure** . Each CLB contains a pair of flip-flops and two independent 4-input function generators. These function generators have a good deal of flexabilty as most combinatorial logic functions need less than four inputs. Thirteen CLB inputs and four CLB outputs provide access to the functional flip-flops. Configurable Logic Blocks implement most of the logic in an FPGA. The principal CLB elements are shown in Figure 1. Two 4-input function generators (F and G) offer unre-stricted versatility. Most combinatorial logic functions need four or fewer inputs. However, a third function generator (H) is provided. The H function generator has three inputs. One or both of these inputs can be the outputs of F and G; the other input(s) are from outside the CLB. The CLB can therefore implement certain functions of up to nine variables, like parity check or expandable-identity comparison of two sets of four inputs.

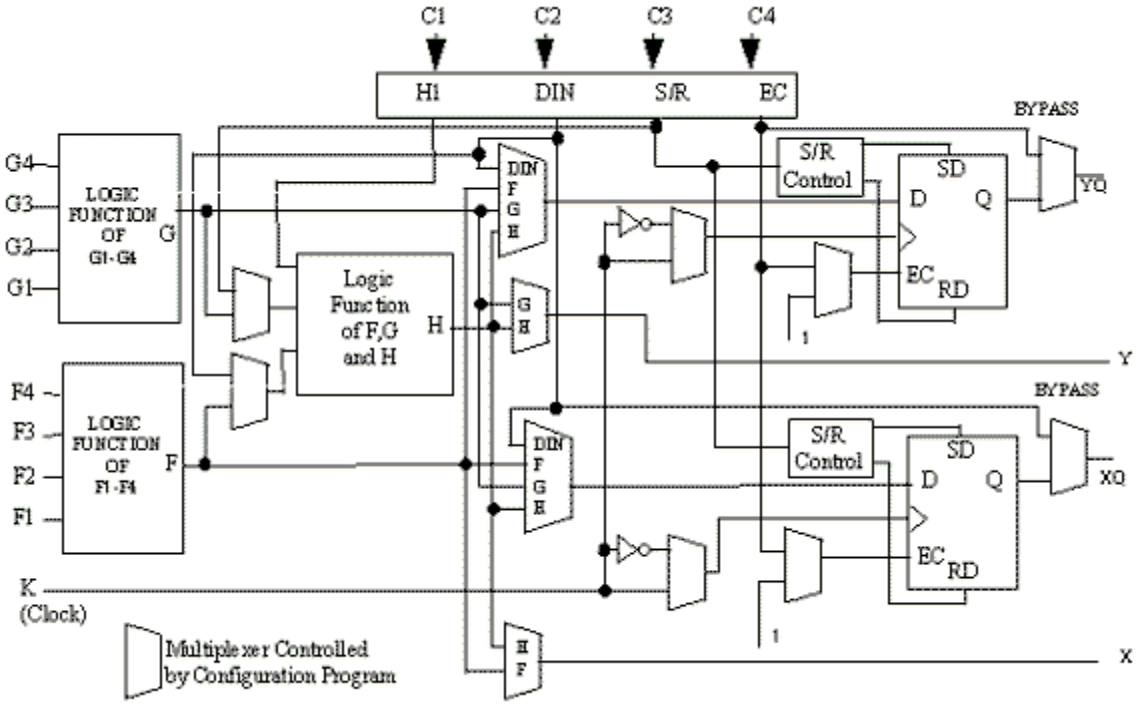


Figure Block Diagram of XC4000 Families Configuration Logic Block (CLB)

Each CLB contains two flip-flops that can be used to store the function generator outputs. However, the flip-flops and function generators can also be used independently. DIN can be used as a direct input to either of the two flip-flops. H1 can drive the other flip-flop through the H function gen-erator. Function generator outputs can also be accessed from outside the CLB, using two outputs independent of the flip-flop outputs. This versatility increases logic density and simplifies routing. Thirteen CLB inputs and four CLB outputs provide access to the function generators and flip-flops. These inputs and outputs connect to the programmable interconnect resources outside the block.

5. 10 HDL LANGUAGE

Advances in semiconductor technology continue to increase the power and complexity of digital systems. To design such systems requires a strong knowledge of Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs), as well as the CAD tools required. Hardware Description Language (HDL) is an essential CAD tool that offers designers

an efficient way for implementing and synthesizing the design on a chip. HDL Programming Fundamentals: VHDL and Verilog teaches students the essentials of HDL and the functionality of the digital components of a system. Unlike other texts, this book covers both IEEE standardized HDL languages: VHDL and Verilog. Both of these languages are widely used in industry and academia and have similar logic, but are different in style and syntax. By learning both languages students will be able to adapt to either one, or implement mixed language environments, which are gaining momentum as they combine the best features of the two languages in the same project. The text starts with the basic concepts of HDL, and covers the key topics such as data flow modeling, behavioral modeling, gate-level modeling, and advanced programming. Several comprehensive projects are included to show HDL in practical application, including examples of digital logic design, computer architecture, modern bioengineering, and simulation.

Digital circuits consist primarily of interconnected transistors. We design and analyze these circuits with the aid of a hierarchical structure: we could, in theory, interpret a central processing unit (CPU) as a vast sea of transistors, but it is much easier to organize transistors into logic gates, logic gates into adders or registers or timing modules, registers into memory banks, and so forth.

To describe digital circuits, textual language is used that is specifically intended to clearly and concisely capture the defining features of digital design.

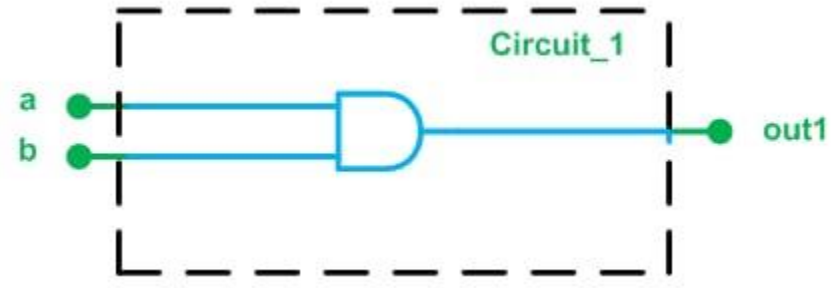
Such languages are called hardware description languages (HDLs).

The most popular hardware description languages are Verilog and VHDL. They are widely used in conjunction with FPGAs, which are digital devices that are specifically designed to facilitate the creation of customized digital circuits.

Hardware description languages allow you to describe a circuit using words and symbols, and then development software can convert that textual description into configuration data that is loaded into the FPGA in order to implement the desired functionality.

```
entity Circuit_1 is
  Port ( a : in  STD_LOGIC;
    b : in STD_LOGIC;
     out1 : out STD_LOGIC);
```

```
  end Circuit_1;
--------------------------------------------------
 architecture Behavioral of Circuit_1 is
 begin
out1 <= ( a and b );
 end Behavioral;
```



NOT gate and half adder HDL Programs

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
 ENTITY not1 IS
 PORT( a : IN STD_LOGIC; b : OUT STD_LOGIC; );
 END not1;
 ARCHITECTURE behavioral OF not1 IS
BEGIN b <= NOT a;
 END behavioral;

entity HALF_ADDER is
port (A, B: in BIT;
 SUM, CARRY: out BIT);
 end HALF_ADDER;
```

5.11 **overview of Spartan 3E**

The XA Spartan®-3E FPGA is the world's lowest cost logic optimized full feature platform of five devices with system gates ranging from 100K to 1.6M gates, and I/Os ranging from 66 to 376 I/Os, with density migration.

XA Spartan-3E FPGA Family Benefits

Industry's lowest total cost

- Large selection of device/package options
- Industry's most comprehensive IP library
- Leading embedded and DSP solutions
- Efficient, cost-effective board designs
- Allows use of fewer components
- Increased system reliability by eliminating external components

Industry's first robust anti-cloning security for low cost FPGAs
Multiple Domain-Optimized Platforms Spartan-3 Generation

The Spartan®-3 Generation of FPGAs offers a choice of five platforms, each delivering a unique cost-optimized balance of programmable logic, connectivity, and dedicated hard IP for your low-cost applications.

- Spartan-3A DSP– DSP Optimized
    - For applications where integrated DSP MACs and expanded memory are required
    - Ideal for designs requiring low cost FPGAs for signal processing applications such as military radio, surveillance cameras, medical imaging, etc.
- Spartan-3AN – Non-volatile

- For applications where non-volatile, system integration, security, large user flash are required
- Ideal for space-critical or secure applications as well as low cost embedded controllers

- Spartan-3A – I/O Optimized
  - For applications where I/O count and capabilities matter more than logic density
  - Ideal for bridging, differential signaling and memory interfacing applications, requiring wide or multiple interfaces and modest processing
- Spartan-3E – Logic Optimized
  - For applications where logic densities matter more than I/O count
  - Ideal for logic integration, DSP co-processing and embedded control, requiring significant processing and narrow or few interfaces
- Spartan-3 – For Highest Density and Pin-Count Applications
  - For applications where both high logic density and high I/O count are important
  - Ideal for highly-integrated data-processing applications

5.12 Virtex II pro FPGA boards- case study

Target Evolvable Hardware

The target EHW is implemented as an OPB slave peripheral module – see Fig. 2. Interfacing with the OPB bus has been simplified by the use of a Xilinx IP Interface core (IPIF). This

provides a simpler interface standard, the Xilinx IPIC, for the user module.Control and configuration of this module are undertaken through register write operations. Genome values are written to registers which are again connected to the configuration inputs of the functional unit array. Registers are also

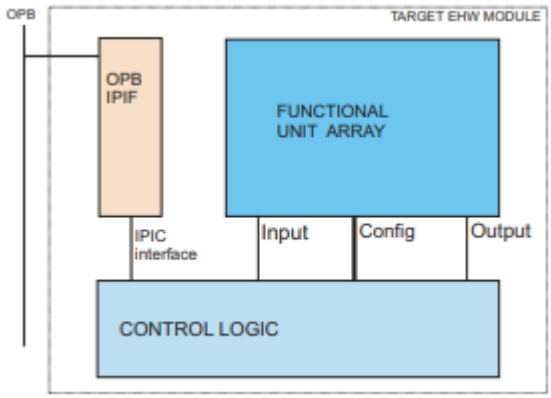


Figure The architecture of target EHW system

Functional unit array

The functional unit array (FUA) is a general structure used for EHW. It is based on the principle that the configuration of the FPGA itself is not changed, but a virtual circuit which is implemented on top of it can be reconfigured. Hence the names ”Virtual FPGA” or ”Virtual Reconfigurable Circuit

Our FUA consists of a fixed-size array of functional units. The array consists of C columns of R units from input to output. Each unit has I inputs, each of which can be connected to any output in the previous column. The unit’s output is a result of any of F functions. The function of each unit and its inputs are configurable. They are determined by evolution, in the way that each individual’s binary genome is sent to the FUA and mapped to the configuration lines. Fitness is then calculated by feeding a number of input vectors on the inputs of the first column, and reading the results from the outputs of the last column. The array is constructed in a pipelined

fashion, that is, registers are connected to the outputs of each layer. Currently, this is not exploited for fitness evaluation. Only one input vector is evaluated at a time.

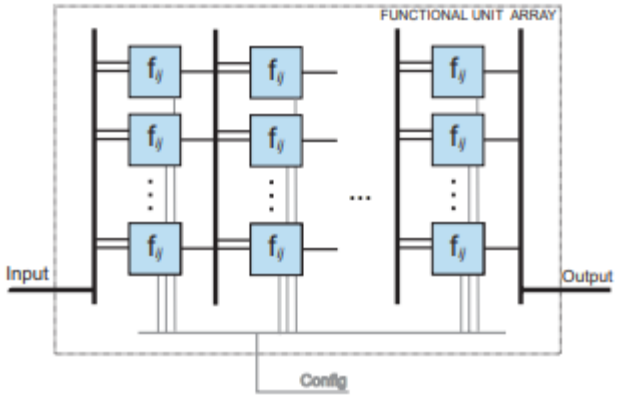


Figure The architecture of the **Functional unit array subsystem.**

#	Description	Function
0	Saturated Add	$O = A + B$, FF if $(A + B) > FF$
1	High Threshold	$O = FF$ if $A > C_2$, else 0
2	Range	$O = FF$ if $C_1 < A < C_2$, else 0
3	Greater	$O = FF$ if $A > B$, else 0
4	Bitwise AND	$O = A$ AND B
5	Bitwise OR	$O = A$ OR B
6	Average	$O = (A + B) >> 1$
7	Half	$O = A >> 1$

Table Functions used by units in the image recognition task. Inputs are A and B, ouput is O. C1 and C2 are constants available to each unit.

2 mark questions and answers.

1. **What is FPGA?**

Field Programmable Gate Arrays (FPGAs) are digital Integrated Circuit that enable the hardware design engineer to program a customized Digital Logic as per requirements.

2. **What is CPLD**?

CPLD is an integrated circuit that helps to implement digital systems whereas FPGA is an integrated circuit designed to be configured by a customer or a designer after manufacturing.

3. **What are the Types of Programmable Logic Devices?**

Programmable Logic Array (PLA)
Programmable Array Logic (PAL)
Generic Array Logic (GAL)

4. **What is CLB** ?
A configurable logic block (CLB) is **the basic repeating logic resource on an FPGA**. When linked together by routing resources, the components in CLBs execute complex logic functions, implement memory functions, and synchronize code on the FPGA.
5. What is PIP in FPGA?
Transmission Gates connects to two wire segments.
0- Wire disconnected
1- Wire connected

6. **Write bout Xilinx 4000 series.**

The Xilinx XC4000E family **includes three major configurable elements: configurable logic blocks (CLBs), input/output blocks, and interconnects**. The CLBs provide the functional elements for constructing user's logic. The IOBs provide the interface between the package pins and internal signal lines.

7. **What is HDL?**

Digital circuits consist primarily of interconnected transistors. We design and analyze these circuits with the aid of a hierarchical structure: we could, in theory, interpret a central processing unit (CPU) as a vast sea of transistors, but it is much easier to organize

transistors into logic gates, logic gates into adders or registers or timing modules, registers into memory banks, and so forth..

Another way to describe digital circuits is to use a textual language that is specifically intended to clearly and concisely capture the defining features of digital design.

Such languages exist, and they are called hardware description languages (HDLs).

8. What do HDLs do?

The most popular hardware description languages are Verilog and VHDL. They are widely used in conjunction with FPGAs, which are digital devices that are specifically designed to facilitate the creation of customized digital circuits.

9. **What is the difference between Programming Languages vs. Hardware Description Languages**?

Statements in HDL code involve parallel operation, whereas programming languages represent sequential operation.

10. What is the use of Spartan®-3 Generation of FPGAs

The Spartan-3 Generation of FPGAs offers a choice of five platforms, each delivering a unique cost-optimized balance of programmable logic, connectivity, and dedicated hard IP for your low-cost applications.

11. What are the HDL languages?

1. VHLD, 2 Verilog

12. **What is the difference between VHLD and Verilog**?

. Both of these languages are widely used in industry and academia and have similar logic, but are different in style and syntax.

13. What is minimum and maximum frequency of dcm in spartan-3 series fpga?

Spartan series dcm's have a minimum frequency of 24 MHZ and a maximum of 248.

14. What are different types of FPGA programming modes?

The modes are Master Parallel, Slave Parallel, Master Serial, Slave Serial, and Boundary Scan.

10 mark questions

1. **Write about some of features of FPGA which are are currently used.**
2. **List out some of synthesizable and non synthesizable constructs**
3. **Draw and explain general structure of fpga.**
4. **Write about FPGA design flow**
5. **Draw a rough diagram of how clock is routed through out FPGA?**
6. **a),What are dcm's? why they are used?**

   b) what is slice, clb, lut?