1. Synchronous: - This type of data transfer is used when device which sends data and devices which receives data are synchronised with the same clock. Works when IO devices and the CPU works with the same speed. IN/OUT instructions are used to transfer data from IO devices to memory and vice versa. Generally used in IO mapped IO scheme, can also be used with memory mapped IO scheme with proper memory read/write instruction. Data is transferred as soon as CPU gives instruction to do so. There is no need to check if the device is ready or not.

2. Asynchronous: - It means at "regular intervals". This type of data transfer scheme is used when speed of the IO device does not match with that of CPU. There is no predictability of timing characteristics. The microprocessor always pings the other device to check whether it's ready or not. During initiation the CPU checks whether device is ready to transfer data, before the actual transfer of data the memory keeps sending signal to IO device. This is called handshaking. The CPU sends initialising signal to device during start and after actual data transfer.

   Interrupt Data transfer scheme: - The program initiates the program and then executes the main program. When IO device is ready to transfer data, the interrupt signal becomes high. The CPU completes the task at hand and then it attends to the IO device. It transfers the data to the stack and then executes a subroutine called ISS (Interrupt Service Subroutine). ISS execution transfers data from IO device to memory and vice versa.

   Direct memory access: - For bulk transfer to or from IO device the above mentioned techniques might prove inefficient. So DMA process is ideal for transferring huge amount of data. The IO device requests the microprocessor by sending a signal. After receiving this request signal the CPU disconnects itself from memory and IO devices by tristating address, data and control bus. The CPU sends the acknowledge signal to IO device. After this data transfer takes place, and on completion IO device withdraws DMA request.

Advancement of architecture of microprocessor

1. Cache memory: - To speed up execution of data, a buffer between the CPU and memory is used. It consists of high speed static ram. Execution speed is equal to microprocessor speed.
2. Pipelining:- This is used to speed up execution of instruction. While the execution unit is working on instructions, the queue in a CPU fetches the next set of instructions. As soon as the working on instruction is over, the next set of instructions are fed into the execution unit. There is no time wasted in fetching instructions. This technique is called pipelining.
3. Multitasking or memory management: - Due to growth in hardware complexity of computers, they were used in time sharing working environment. That means a fixed amount of time is allocated to different programmes. To achieve relocatablity segmented scheme is used.
4. Virtual memory system:- In this scheme the complete program is divided into several pgs. and stored in hard disk. At same time the main memory is divided into small pages.

PIN NO - 23

TYPE – INPUT

FUNCTION:

The signal is used in wait instruction before execution the instruction microprocessor check the (TEST)' pin status.

If TEST =1, then microprocessor will not enter into wait state , that is - execution will continue.

If TEST = 0, then the microprocessor will enter into wait state.

NMI(NON MARKABLE INTERRUPT):

PIN- 17

TYPE: - INPUT

FUNCTION:

This signal cannot be makeable internally by software. this is a edge triggered signal which cause a type 2 interrupt when signal is active high interrupt service is vector to via an interrupt vector.

RESET:

PIN NO.- 21

TYPE- INPUT

FUNCTION:-

When this pin is high it immediately reset the microprocessor, When the pin status is high immediately the segment count is FFFFH and so the base address at that moment is $FFFF_0H$

3. REGISTER ORGANISATION:-

The INTEL 8086 contains the following register

(a) General purpose register

(b) Pointer and index register

(c) Segment register

(d) Instruction pointer

(e) Status Flags

(a) General Purpose register:-

(i) The AX, BX, CX and DX are the general purpose 16-bit register.

(ii) AX is used as 16-bit accumulator.

(iii) AX ->AH (For higher 8-bit)

➔ AL (For lower 8-bit)

MODULE 3

Instruction Set of 8086

The 8086 instructions are categorized into the following main types.

i. Data Copy / Transfer Instructions
ii. Arithmetic and Logical Instructions
iii. Branch Instructions
iv. Loop Instructions
v. Machine Control Instructions
vi. Flag Manipulation Instructions
vii. Shift and Rotate Instructions
viii. String Instructions

Data Copy / Transfer Instructions :

MOV :

This instruction copies a word or a byte of data from some source to a destination. The destination can be a register or a memory location. The source can be a register, a memory location, or an immediate number.

MOV AX,BX
MOV AX,5000H
MOV AX,[SI]
MOV AX,[2000H]
MOV AX,50H[BX]
MOV [734AH],BX
MOV DS,CX
MOV CL,[357AH]
Direct loading of the segment registers with immediate data is not permitted.

3

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.

E.g. PUSH AX
• PUSH DS
• PUSH [5000H]

Fig. 2 Push Data to stack memory

POP : Pop from Sack

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.

The stack pointer is incremented by 2

Eg. POP AX
POP DS
POP [5000H]

Fig 3 Popping Register Content from Stack Memory

XCHG : Exchange byte or word

This instruction exchange the contents of the specified source and destination operands

Eg. XCHG [5000H], AX

multiplication and comparing two values.

ADD :

The add instruction adds the contents of the source operand to the destination operand.

Eg. ADD AX, 0100H
ADD AX, BX
ADD AX, [SI]
ADD AX, [5000H]
ADD [5000H], 0100H
ADD 0100H

ADC : Add with Carry

This instruction performs the same operation as ADD instruction, but adds the carry flag to the result.

Eg. ADC 0100H
ADC AX, BX
ADC AX, [SI]
ADC AX, [5000]
ADC [5000], 0100H
6

SUB : Subtract

The subtract instruction subtracts the source operand from the destination operand and the result is left in the destination operand.

Eg. SUB AX, 0100H
SUB AX, BX
SUB AX, [5000H]
SUB [5000H], 0100H

SBB : Subtract with Borrow

The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand

Eg. SBB AX, 0100H
SBB AX, BX
SBB AX, [5000H]
SBB [5000H], 0100H

INC : Increment

This instruction increases the contents of the specified Register or memory location by 1. Immediate data cannot be operand of this instruction.

Eg. INC AX
INC [BX]
INC [5000H]

DEC : Decrement

The decrement instruction subtracts 1 from the contents of the specified register or memory location.

Eg. DEC AX
DEC [5000H]

NEG : Negate

The negate instruction forms 2's complement of the specified destination in the instruction. The destination can be a register or a memory location. This instruction can be implemented by inverting each bit and adding 1 to it.

Eg. NEG AL
AL = 0011 0101 35H Replace number in AL with its 2's complement

Eg. OR AX, 0008H
OR AX, BX

NOT : Logical Invert

This instruction complements the contents of an operand register or a memory location, bit by bit.
Eg. NOT AX
NOT [5000H]

XOR : Logical Exclusive OR

This instruction bit by bit XORs the source operand that may be an immediate , register or a memory location to the destination operand that may a register or a memory location. The result is stored in the destination operand.
Eg. XOR AX, 0098H
XOR AX, BX

TEST : Logical Compare Instruction

The TEST instruction performs a bit by bit logical AND operation on the two operands. The result of this ANDing operation is not available for further use, but flags are affected.
Eg. TEST AX, BX
TEST [0500], 06H
10

SAL/SHL : SAL / SHL destination, count.

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination to the left and 0 is stored at LSB position. The MSB is shifted into the carry flag. The destination can be a byte or a word.
It can be in a register or in a memory location. The number of shifts is indicated by count.
Eg. SAL CX, 1
SAL AX, CL

SHR : SHR destination, count

This instruction shifts each bit in the specified destination to the right and 0 is stored at MSB position. The LSB is shifted into the carry flag. The destination can be a byte or a word.
It can be a register or in a memory location. The number of shifts is indicated by count.
Eg. SHR CX, 1
MOV CL, 05H
SHR AX, CL

SAR : SAR destination, count

This instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position. The LSB will be shifted into CF.
Eg. SAR BL, 1
MOV CL, 04H
SAR DX, CL

ROL Instruction : ROL destination, count

This instruction rotates all bits in a specified byte or word to the *left* some number of bit positions. MSB is placed as a new LSB and a new CF.
Eg. ROL CX, 1
MOV CL, 03H
ROL BL, CL

ROR Instruction : ROR destination, count

CMPS : Compare String Byte or String Word
The CMPS instruction can be used to compare two strings of byte or words. The length of the string must be stored in the register CX. If both the byte or word strings are equal, zero Flag is set.
The REP instruction Prefix is used to repeat the operation till CX (counter) becomes zero or the condition specified by the REP Prefix is False.
SCAN : Scan String Byte or String Word
This instruction scans a string of bytes or words for an operand byte or word specified in the register AL or AX. The String is pointed to by ES:DI register pair. The length of the string s stored in CX. The DF controls the mode for scanning of the string. Whenever a match to the specified operand, is found in the string, execution stops and the zero Flag is set. If no match is found, the zero flag is reset.
LODS : Load String Byte or String Word
The LODS instruction loads the AL / AX register by the content of a string pointed to by DS : SI register pair. The SI is modified automatically depending upon DF, If it is a byte transfer (LODSB), the SI is modified by one and if it is a word transfer (LODSW), the SI is modified by two. No other Flags are affected by this instruction.
16
STOS : Store String Byte or String Word
The STOS instruction Stores the AL / AX register contents to a location in the string pointer by ES : DI register pair. The DI is modified accordingly, No Flags are affected by this instruction.
The direction Flag controls the String instruction execution. The source index SI and Destination Index DI are modified after each iteration automatically. If DF=1, then the execution follows autodecrement mode, SI and DI are decremented automatically after each iteration. If DF=0, then the execution follows autoincrement mode. In this mode, SI and DI are incremented automatically after each iteration.
Flag Manipulation and Processor Control Instructions
These instructions control the functioning of the available hardware inside the processor chip. These instructions are categorized into two types:
1. Flag Manipulation instructions.
Machine Control instructions.
Flag Manipulation instructions
The Flag manipulation instructions directly modify some of the Flags of 8086.
i. CLC – Clear Carry Flag.
ii. CMC – Complement Carry Flag.
iii. STC – Set Carry Flag.
iv. CLD – Clear Direction Flag.
v. STD – Set Direction Flag.
vi. CLI – Clear Interrupt Flag.
vii. STI – Set Interrupt Flag.
Machine Control instructions
The Machine control instructions control the bus usage and execution
i. WAIT – Wait for Test input pin to go low.
ii. HLT – Halt the process.
iii. NOP – No operation.
iv. ESC – Escape to external device like NDP
v. LOCK – Bus lock instruction prefix.
17
Addressing Modes
Addressing modes of 8086

when memory is accessed PA is computed from BX and DS when the stack is accessed PA is computed from BP and SS.
Example : MOV AL, START [BX]
or
MOV AL, [START + BX]
based mode
EA : [START] + [BX]
PA : [DS] + [EA]
The 8 bit content of this memory location is moved to AL.
20
Indexed addressing mode:
CS
PA = DS SI
SS : or + 8 or 16bit displacement
ES DI
Example : MOV BH, START [SI]
PA : [SART] + [SI] + [DS]
The content of this memory is moved into BH.
Based Indexed addressing mode:
CS
PA = DS BX SI
SS : or + or + 8 or 16bit displacement
ES BP DI
Example : MOV ALPHA [SI] [BX], CL
If [BX] = 0200, ALPHA – 08, [SI] = 1000 H and [DS] = 3000
Physical address (PA) = 31208
8 bit content of CL is moved to 31208 memory address
String addressing mode:
The string instructions automatically assume SI to point to the first byte or word of the source operand and DI to point to the first byte or word of the destination operand. The contents of SI and DI are automatically incremented (by clearing DF to 0 by CLD instruction) to point to the next byte or word.
Example : MOV S BYTE
If [DF] = 0, [DS] = 2000 H, [SI] = 0500,
[ES] = 4000, [DI] = 0300
Source address : 20500, assume it contains 38
PA : [DS] + [SI]
Destination address : [ES] + [DI] = 40300, assume it contains 45
After executing MOV S BYTE,
[40300] = 38
[SI] = 0501 incremented
[DI] = 0301
C. I/O mode (direct) :
Port number is an 8 bit immediate operand.
Example : OUT 05 H, AL
Outputs [AL] to 8 bit port 05 H
I/O mode (indirect):
The port number is taken from DX.
Example 1 : INAL, DX
21
OR

Example : Block move program using the move string instruction
MOV AX, DATA SEG ADDR
MOV DS, AX
MOV ES, AX
MOV SI, BLK 1 ADDR
MOV DI, BLK 2 ADDR
32
MOV CK, N
CDF ; DF=0
NEXT : MOV SB
LOOP NEXT
HLT
Load and store strings :(LOD SB/LOD SW and STO SB/STO SW)
LOD SB: Loads a byte from a string in memory into AL. The address in SI is used relative to DS to determine the address of the memory location of the string element.
(AL) ← [(DS) + (SI)]
(SI) ← (SI) + 1
LOD SW : The word string element at the physical address derived from DS and SI is to be loaded into AX. SI is automatically incremented by
(AX) ← [(DS) + (SI)]
(SI) ← (SI) + 2
STO SB : Stores a byte from AL into a string location in memory. This time the contents of ES and DI are used to form the address of the storage location in memory
[(ES) + (DI)] ← (AL)
(DI) ← (DI) + 1
STO SW :[(ES) + (DI)] ← (AX)
(DI) ← (DI) + 2
Mnemonic Meaning Format Operation Flags affected
MOV SB
Move
String
Byte
MOV
SB
((ES)+(DI))←((DS)+(SI))
(SI)←(SI) m 1
(DI) ← m 1
None
MOV SW
Move
String
Word
MOV
SW
((ES)+(DI))←((DS)+(SI))
((ES)+(DI)+1)←(DS)+(SI)+1)
(SI) ← (SI) m 2
(DI) ← (DI) m 2
None
LOD SB /

LOD SW
Load
String
LOD
SB/
LOD
SW
(AL) or (AX) ←((DS)+(SI))
(SI)←(SI) m 1 or 2
None
33
STOSB/
STOSW
Store
String
STOSB/
STOSW
((ES)+(DI))←(AL) or (AX)
(DI) ← (DI) 71 or 2
None
Example : Clearing a block of memory with a STOSB operation.
MOV AX, 0
MOV DS, AX
MOV ES, AX
MOV DI, A000
MOV CX, OF
CDF
AGAIN : STO SB
LOOP NE AGAIN
NEXT :
Clear A000 to A00F to 0016

Repeat String : REP

The basic string operations must be repeated to process arrays of data. This is done by inserting a repeat prefix before the instruction that is to be repeated.

Prefix REP causes the basic string operation to be repeated until the contents of register CX become equal to zero. Each time the instruction is executed, it causes CX to be tested for zero, if CX is found to be nonzero it is decremented by 1 and the basic string operation is repeated.

Example : Clearing a block of memory by repeating STOSB
MOV AX, 0
MOV ES, AX
MOV DI, A000
MOV CX, OF
CDF
REP STOSB
NEXT:

The prefixes REPE and REPZ stand for same function. They are meant for use with the CMPS and SCAS instructions. With REPE/REPZ the basic compare or scan operation

can be repeated as long as both the contents of CX are not equal to zero and zero flag is

1.
REPNE and REPNZ works similarly to REPE/REPZ except that now the operation is repeated as long as CX≠0 and ZF=0. Comparison or scanning is to be performed as long as the string elements are unequal (ZF=0) and the end of the string is not yet found (CX≠0).

Prefix Used with Meaning

REP
MOVS
STOS
Repeat while not end of
string CX≠0

REPE/ REPZ
CMPS
SCAS
CX≠0 & ZF=1

REPNE/REPNZ
CMPS
SCAS
CX≠0 & ZF=0

Example : CLD ; DF =0
MOV AX, DATA SEGMENT ADDR
MOV DS, AX
MOV AX, EXTRA SEGMENT ADDR
MOV ES, AX
MOV CX, 20
MOV SI, OFFSET MASTER
MOV DI, OFFSET COPY
REP MOVSB

Moves a block of 32 consecutive bytes from the block of memory locations starting at offset address MASTER with respect to the current data segment (DS) to a block of locations starting at offset address copy with respect to the current extra segment (ES).

Auto Indexing for String Instructions :

SI & DI addresses are either automatically incremented or decremented based on the setting of the direction flag DF.
When CLD (Clear Direction Flag) is executed DF=0 permits auto increment by 1.
When STD (Set Direction Flag) is executed DF=1 permits auto decrement by 1.
35

Mnemonic Meaning Format Operation Flags affected

CLD Clear DF CLD (DF) ← 0 DF
STD Set DF STD (DF) ← 1 DF

1. LDS Instruction:

LDS register, memory (Loads register and DS with words from memory)
This instruction copies a word from two memory locations into the register specified in the instruction. It then copies a word from the next two memory locations into the DS register. LDS is useful for pointing SI and DS at the start of the string before using one of the string instructions. LDS affects no flags.
Example 1 :LDS BX [1234]
Copy contents of memory at displacement 1234 in DS to BL. Contents of 1235H to BH. Copy contents at displacement of 1236H and 1237H is DS to DS register.
Example 2 : LDS, SI String – Pointer

(SI) ← [String Pointer]
(DS) ← [String Pointer +2]
DS, SI now points at start and desired string

LEA Instruction :

Load Effective Address (LEA register, source)
This instruction determines the offset of the variable or memory location named as the source and puts this offset in the indicated 16 bit register.
LEA will not affect the flags.
Examples :
LEA BX, PRICES
Load BX with offset and PRICES in DS
LEA BP, SS : STACK TOP
Load BP with offset of stack-top in SS
LEA CX, [BX] [DI]
Loads CX with EA : (BX) + (DI)
36

3. LES instruction :

LES register, memory
Example 1: LES BX, [789A H]
(BX) ← [789A] in DS
(ES) ← [789C] in DS
Example 2 : LES DI, [BX]
(DI) ← [BX] in DS
(ES) ← [BX+2] in DS

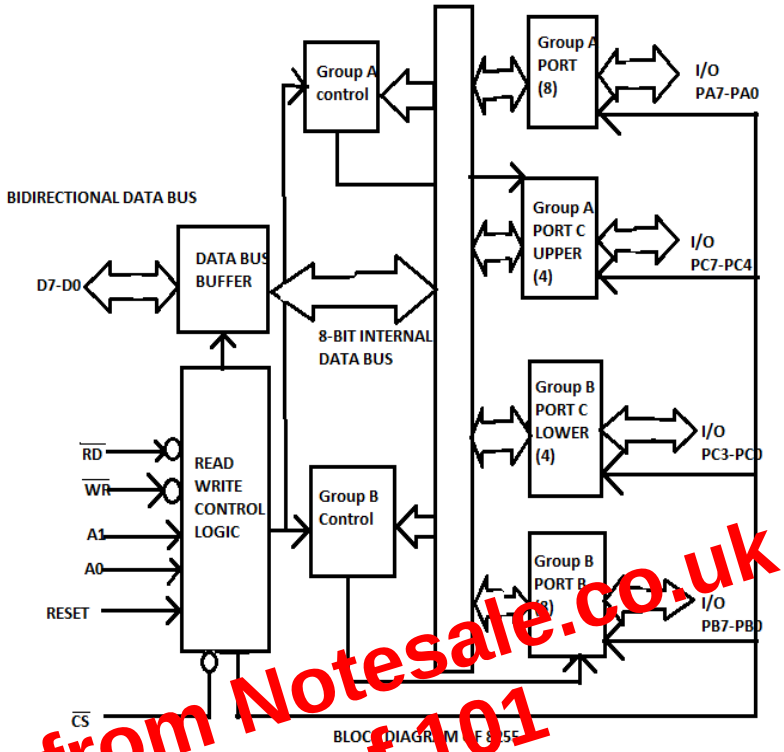
PROGRAMMABLE PERIPHERAL INTERFACE (8255)

The programmable peripheral interface i s a low cost interfacing circuit used in many applications.its function is to perform input output operation.it contains 3 I/O ports, 24 I/O pins which can be programmed in three different modes.The various I/O operations can be performed by writing instructions in its internal control word register. Along basic I/O operation it also performs time delay generation counting generating signals and interrupts.

Features

1. High speed and low speed consumptions due to CMOS technology.
2. It is PPI device .
3. Power supply ranges 3 volts to 6 volts.
4. PPI has 24 I/O programmable pins in groups of 12 pins which are arranged as 3 8 bit ports (PORT A,PORT B,PORT C).
5. it is used for the interface to keyboard and parallel to printer port.

BLOCK DIAGRAM OF 8255 AND ARCHITECTURE



DATA BUS BUFFER

The 8 bit bidirectional data bus connected to data bus of the micro processor. The direction of the data bus are decided by the read and write control signals . in read operation it transmit data to the system bus and in write operation it receives data from system bus.

READ /WRITE CONTRO LOGIC

the block function is to accepts inputs from system control bus and system bus. The control signal $\overline{RD}$ and$\overline{WR}$ $\overline{CS}$ and the address signal used as A1,A0.

A1 and A0 are connected to address lines A2 and A1 resp. Of system address lines .if $\overline{CS}$=08255 is selected else rejected.

GROUP A AND GROUP B CONTOL

Group A consist of PORT A AND PORT C(upper).Group consist of PORT B and PORT C lower. Each grpup consist of 12 pins .selection of PORT bits are done by mode operation

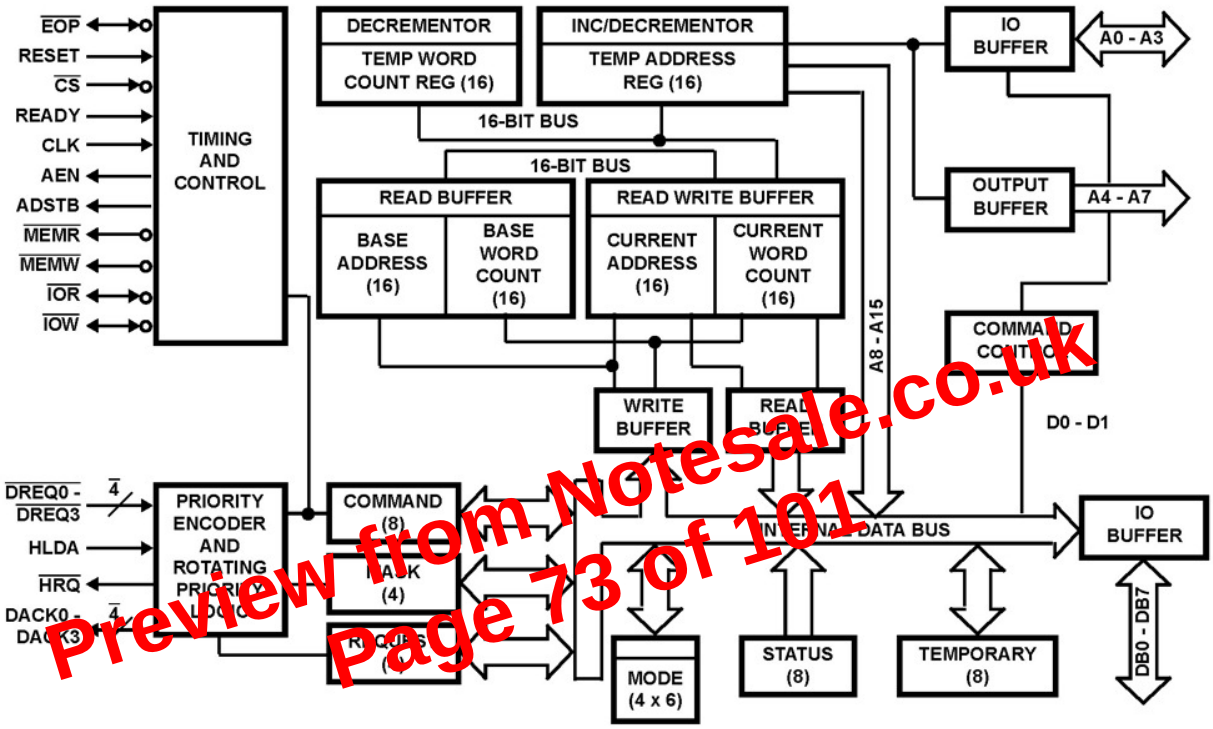
PORT A,PORT B AND PORT C

Each port consist of 8 bit data input buffer. The function of these ports are decided by control bit pattern control word register. PORT C is divided into PC (upper) and PC (lower), used as simple input or output, hand shake signals and status signals .

A1	A0	PORT/register selection

channels and may be expanded to any number of channels by cascading additional controller chips. Thethree basic transfer modes allow programmability of the types of DMA service by the user. Each channel canbe individually programmed to Autoinitialize to its original condition following an End of Process (EOP). Eachchannel has a full 64K address and word count capability.

BLOCK DIAGRAM OF 8237



REGISTER ORGANISATION OF 8237

1.CURRENT ADDRESS REGISTER:

Each of the four channels of 8237 has a 16 –bit current address register that hold the current memory address. The address is automatically increamented or decremented after each transfer and the resulting address value is again stored in the current address register.

2.CURRENT WORD REGISTER:

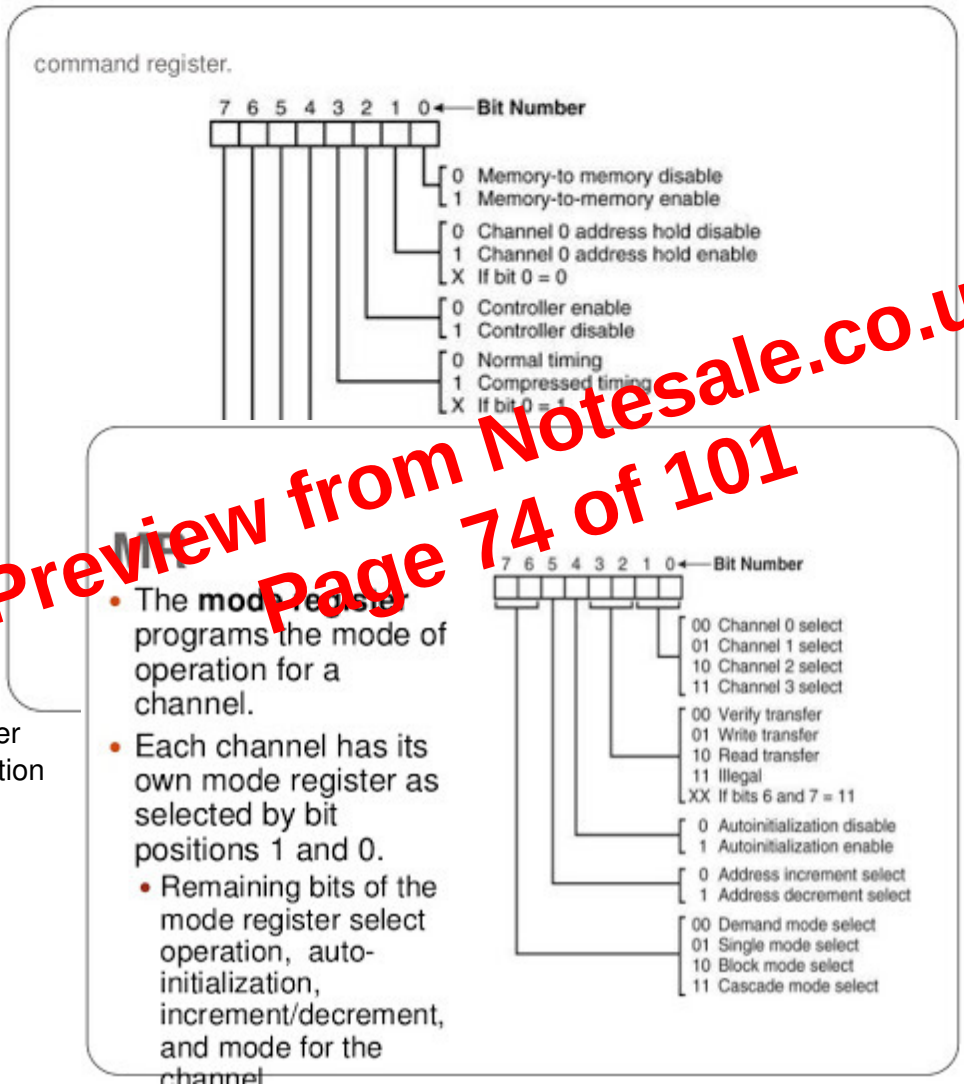
Each channel has 16 –bit current word register that hold the no of databyte transfers to be carried out.The word count is decremented after each transfer and the new value is again stored in control word register.When count becomes zero an EOP signal will be generated.after EOP this may be reinitialized using autoinitialised command.

3.BASE ADRESS AND BASE WORD COUNT REGISTER:

Each channel has a pair of these register.These are automatically written along with the current register.These cannot be read by the CPU.The contents of these registers are used for auto initialization.

4.COMMAND REGISTER:

This 8-bit register controls the entire operation of 8237.This can be progammed by the CPU and cleared by a reset operation.



Each

an 8-

0 and

4

to be

2 and

the

DMA

4

wheather

intialization

or not.

5.MODE REGISTER:

In the DMA channel has bit mode register.Bits 1 determine which of the channel is written.Bits 3 indicates type of transfer.Bit indicates auto is selected

The bit configuration of mode instruction is shown in Figures 2 and 3. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

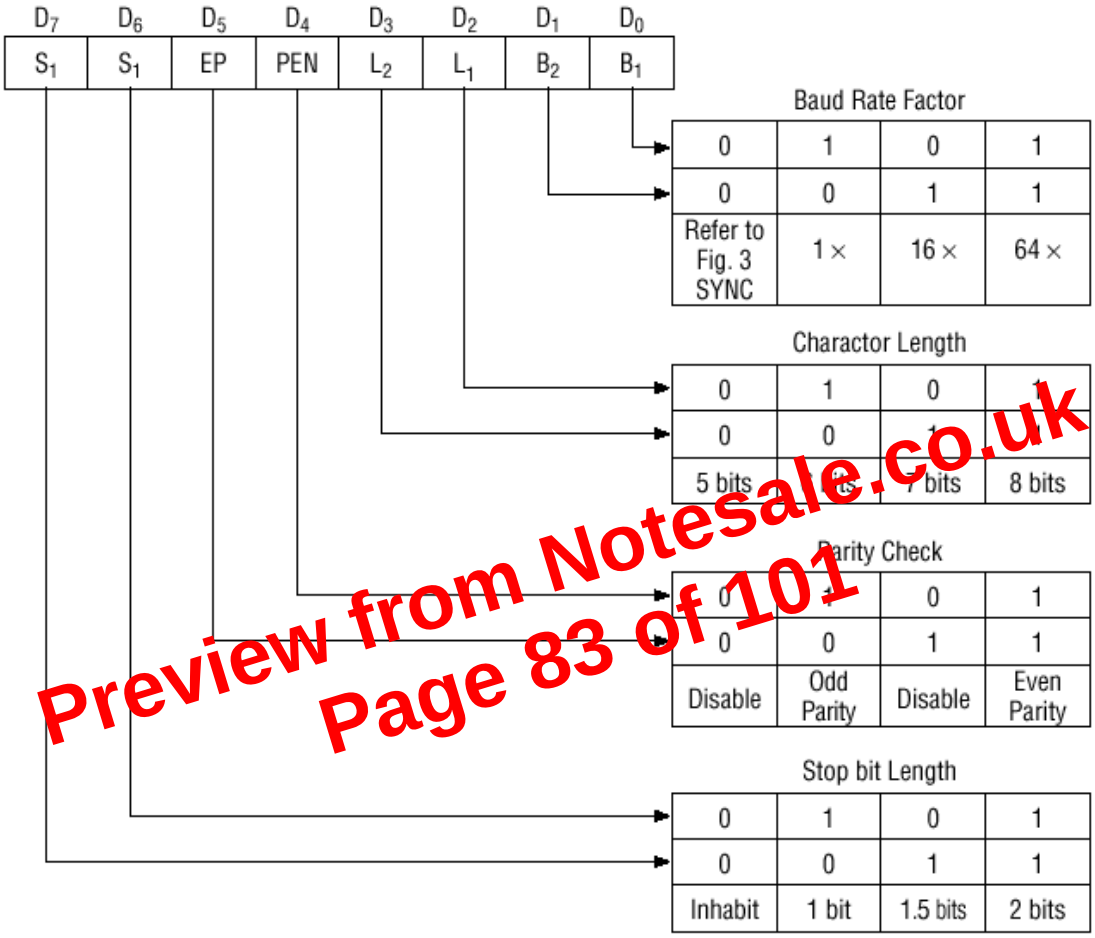


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

B.)FIFO STATUS WORD

It is used in keyboard and strobbed input mode to indicate error. If FIFO is full and write is attempted then overrun error occurs .If FIFO is empty and read is attempted then underrun error occurs.

D7	D6	D5	D4	D3	D2	D1	D0
Du	S/E	O	U	F	N	N	N

Du-display RAM unavailable due to clearing operation

S/E-sensor closure or error flag for multiple closures

O-overrun error, if O=1

U-underrun error if U=1

FIFO full If F=1

D0, D1, D2—number of character that are available for reading from FIFO

INTERFACING WITH 8086