AS-4153

B. Tech (fifth Semester) Examination 2013

Microprocessor and Interfaces

MODEL ANSWER

1 Choose the correct answer :

(i) b Only 1 and 2 is correct

(ii) 1KB

(iii) No instruction is there

(iv) No instruction

(v) $\overline{CS}$

(vi) Write control word

(vii) 1, 2, 3, 4

(viii) 16

(ix) pushes only IP register in stack

(x) 37 , CF = 1

Unit I

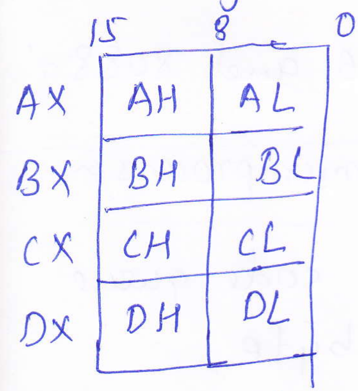
2(i) Architectural difference between 8086 and 8088:-

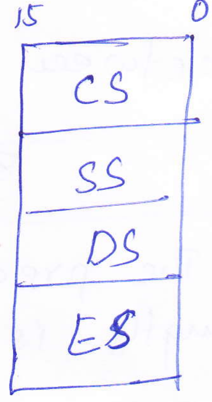
8086 microprocessor	8088 microprocessor
1) The precoded code queue length is 6 bytes.	1) The precoded code queue length is 4 byte
2) 8086 is available in three clock speed 5MHz, 8MHz and 10MHz	2) 8088 is available in two clock speed 5MHz and 8MHz.
3) The 8086 bus interface unit will fetch a byte from memory to load the queue each time, if at least 2byte is free.	3) The 8088 bus interface unit will fetch a byte from memory to load the queue each time, if at least 1 byte is free.
4) The memory space of 8086 is organized as banks.	4) The memory space of 8086 is implemented as single bank 1M *8 memory bank.
5) The 8086 can fetch 16-bit data in two consecutive bus cycle.	5) The 8088 can fetch 16-bit data in 4 clock cycle.

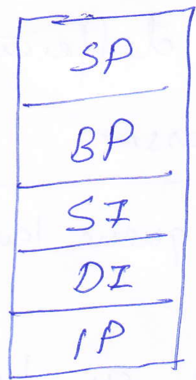
Signal difference between 8086 and 8088

8086	8088
1) A8-A15 is multiplexed with data bus.	1) A8-A15 is already latched all time valid address bus
2) $\overline{BHE}$ signal is there to transfer data over higher data bus.	2) $\overline{BHE}$ has no meaning as the data bus is of 8bit
3) ~~IO/M~~ M/$\overline{IO}$ pin is available to identify memory and IO operation	3) IO/$\overline{M}$ pin is available to identify memory and IO operation.
4) SSo pin is not required.	4) SSo pin is introduced in 8088

2 (ii) Register Organisation of 8086







~~Seg~~ General data Register :- The AX, BX, CX and DX are the general purpose 16-bit register, which can be used either to carry 8 bit data using AH register or AL register, or can directly carry 16 bit data using AX register.

AX is used as 16 bit accumulator. This is the most important general purpose register having multiple function

BX register is used as an offset storage for forming physical address in case of certain addressing mode.

CX register is used as default register in case of string and loop instruction.

DX register is used as an implicit operand or destination in case of few instruction.

Segment Register :-

Code Segment :- The code segment register is used for addressing a memory location in the code segment of

<u>Data Segment</u> -: This register points to the data segment of the memory, where the data is resided.

<u>Extra Segment</u> -: This register refer a segment which essentially is another data segment of the memory.

<u>Stack Segment</u> -: This register is used for addressing stack segment memory i.e. memory which is used to store stack data.

Above all ~~Sta~~ Segment Register is used to hold the upper 16-bit of starting address for each of the segment.

<u>Pointer and Index Register</u> -:

<u>SP register</u> -: The stack pointer is used to hold offset address of the data stored at the top of the stack segment.

<u>BP register</u> -: Base Pointer contains an assumed offset from the SS register. Often used by subroutine to locate variable that were passed on the stack by a calling program.

<u>SI register</u> -: Source Register is used to hold offset address of the source data segment while executing string instruction.

<u>DI register</u> -: Destination Register is used to hold the offset address of the destination data in extra segment; while executing the string instruction.

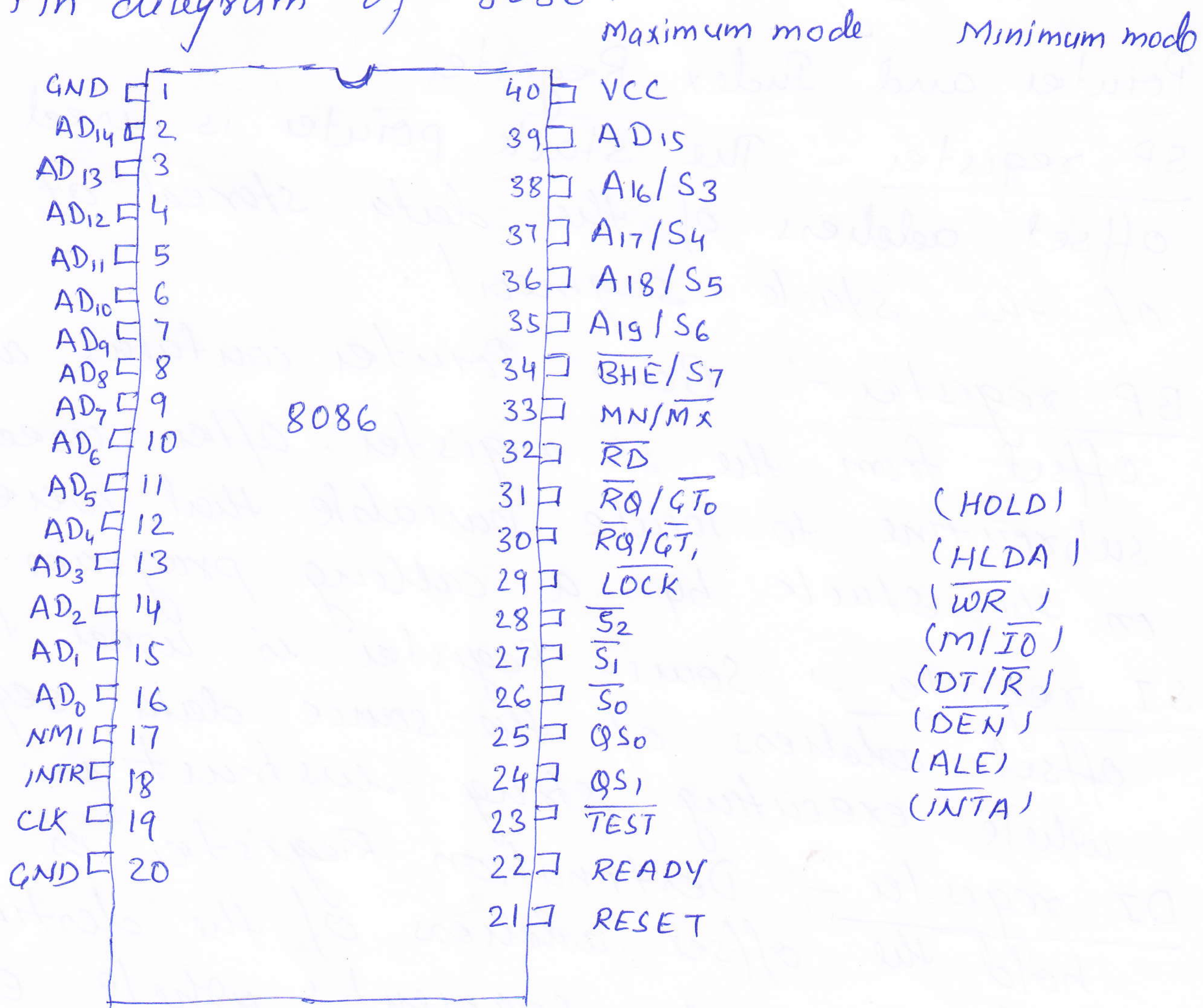
IP register :- Instruction pointer contains the offset of the next instruction to be executed in code segment.

Flag Register :- ~~Individual~~ It is a 16 bit register show status of CPU or result of arithematic operations.

- Control flags (Direction, Interrupt, Trap)
- Status flag (Carry, Overflow, Sign, Zero, Auxiliary Carry, Parity)

— x —

2 (iii) Pin diagram of 8086.



3(i)

ASSUME :- The ASSUME directive is used to inform the assembler, the names of the logicals segment to be assumed for different segment used in the program. For example, the code segment may be given the name CODE, data segment may be given the name DATA etc.

Exe :- ASSUME CS: CODE, DS: DATA

The above statement directs the assembler that the machine code are available in a segment named CODE, the CS register is to be loaded with the address allotted by the operating system.

SEGMENT :- The SEGMENT directive marks the starting of a logical segment. The started segment is also assigned a name, i.e. label, by this statement. The SEGMENT and ENDS directive must bracket each logical segment of a program.

Exe :

```
START   SEGMENT   GLOBAL ; Start of segment
                         ; named START
           :
START   ENDS
```

OFFSET : When the assembler comes across the OFFSET operator along with a label, it first computes the 16-bit displacement of particular label, and replaces the string 'OFFSET LABEL' by the computed displacement.

Exe : MOV SI, OFFSET LIST

The above statement will calculate the 16-bit displacement of 'LIST' and will assign to SI register

GROUP :- This directive is used to form logical ~~segment~~ group of segment with similar purpose or type. This directive is used to inform the assembler to form a logical group of the following segment names.

Exe : PROGRAM GROUP CODE, DATA, STACK

The above statement directs loader/linker to prepare an EXE file such that CODE, DATA and STACK segment must lie within 64kbyte memory segment.

3 (ii) Assembly program to perform one byte BCD addition :-

```
MOV AL, 23H
MOV BL, 34H
ADD AL, BL
DAA
ADC AH, 00H
INT 3
```

3(iii)

(a) ADC :- This instruction perform addition of an immediate data or content of a memory location specified in the instruction or a register to the contents of another register with carry flag bit. All the condition code flag are affected by this instruction.

Exe :-
```
ADC AX, BX
ADC AX, [5000H]
ADC [5000H], 0100H
```

(b) NEG :- The negate instruction forms 2's complement of the specified destination in the instruction. For obtaining 2's complement, it subtracts the content of destination from zero. The result is stored back in the destination. This instruction affects all the condition code flags.

Exe :

```
MOV AL, 05 H
NEG AL
```

(c) CMP :- This instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison it substracts the source operand from the destination operand but does not store the result anywhere. If both of the operand are equal, zero flag is set. If the source operand is greater than the destination operand carry flag is set or else carry flag is reset.

Exe :-

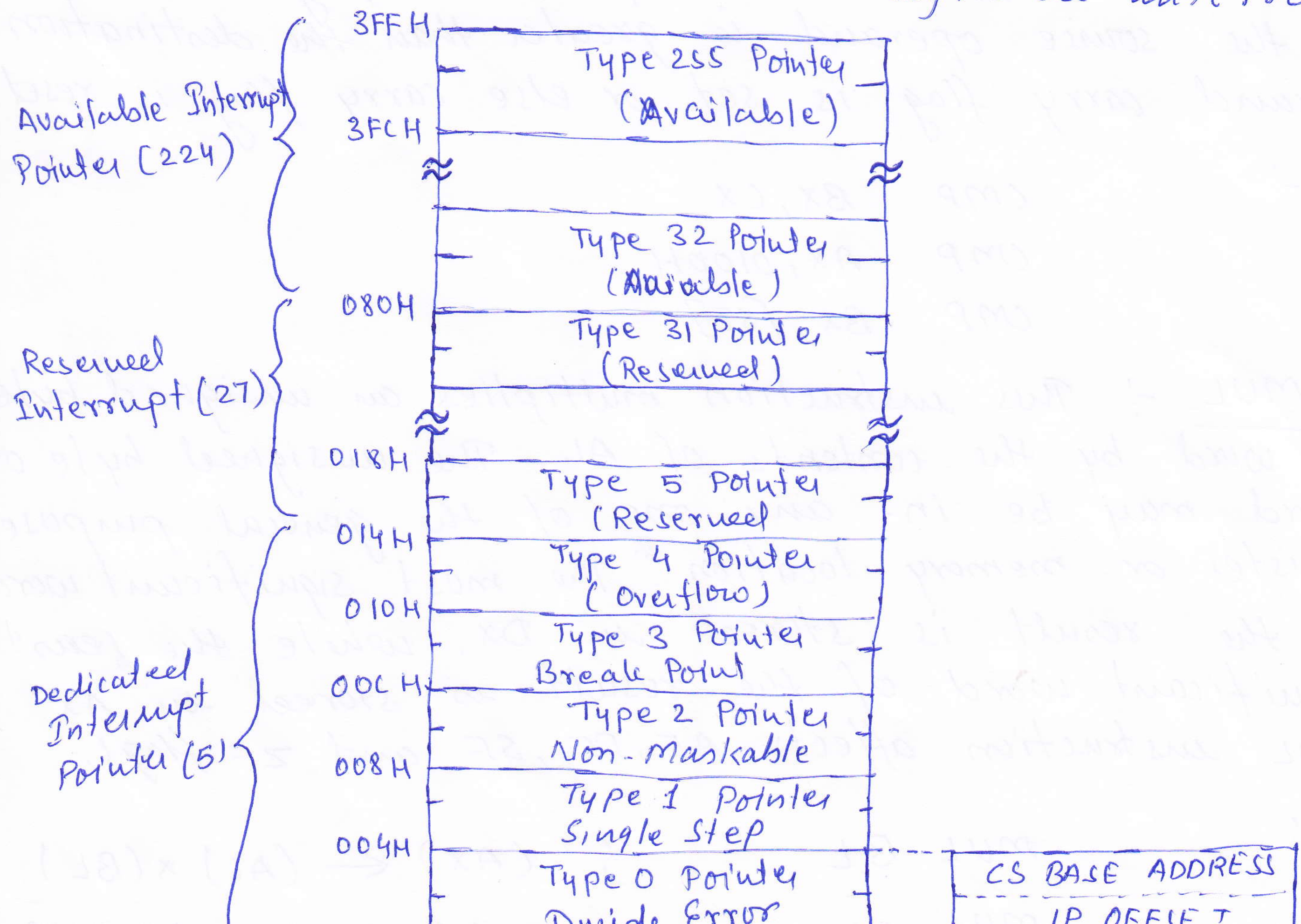
```
CMP  BX, CX
CMP  AX, 0100H
CMP  BX, [SI]
```

(d) MUL :- 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 register or memory location. The most significant word of the result is stored in DX, while the least significant word of the result is stored in AX. MUL instruction affects AF, PF, SF and ZF flags.

Exe :

```
MUL BL        ; (AX) ← (AL) × (BL)
```

4(i) In an 8086 system the first 1 kbyte of memory from 00000H to 003FFH is reserved for storing the starting addresses of interrupt service routine. This block of memory is often called the interrupt vector table. Since 4 bytes are required to store the CS and IP values for each interrupt service procedure, the table can hold the starting address for 256 interrupt service routine. Each interrupt type is given a number between 0 to 255 and the address of each interrupt is found by multiplying the type by 4. Only first five types have explicit definition such as divide by zero, The next 27 interrupt types from 5 to 31 are reserved for future use. The upper 224 interrupt type from 3 to 255 are available for hardware and software interrupts.



4 (ii). Macro is a group of instruction. The macro assembler generates the code in the program each time where the macro is called. A MACRO can be defined anywhere in a program using the directive MACRO and ENDM. The label prior to MACRO is the macro name which should be used in the actual Program. The ENDM directive marks the end of the instruction

Syntax :-

```
Macro Name   MACRO
               :
               :
             ENDM
```

Example:

```
DISPLAY  MACRO
         MOV AX, 0010H
         MOV DS, AX
         MOV DX, OFFSET MSG
         MOV AH, 09H
         ENDM
```

Passing Parameter in MACRO :- Using parameter in a definition, the programmer specifies the parameter of the macro.

```
DISPLAY  MACRO  MSG
         MOV AX, 0010H
         MOV DS, AX
         MOV DX, OFFSET MSG
         MOV AH, 09H
         INT 21H
         ENDM
```

~~This~~ The above parameter MSG can be replaced by MSG1 or MSG2 while calling the macro

```
        :
        :
DISPLAY MSG1
        :
        :
DISPLAY MSG2
        :
        :
MSG1 DB 0AH,0DH "Program"
MSG2 DB 0AH,0DH "Process"
```

4(iii) String Manipulation instruction of 8086 -:

(a) MOVSB/MOVSW : Move String Byte or String Word

These instruction copy a byte or word from a location in the data segment to a location in the extra segment. The starting byte of source string is located in the memory location whose address may be computed using SI and DS content. The starting address of destination location is computed using DI and ES.

Exe -:

```
MOV AX,0000H
MOV DS,AX
MOV ES,AX
MOV SI,2000H
MOV DI,2400H
MOV CX,04H
REP MOVSB
```

(b) CMPS : Compare String Byte or String Word

The CMPS instruction can be used to compare two strings of bytes or words. The length of string must be stored in the register CX. If both the byte or word

Exe :-

```
MOV AX, 5000H
MOV DS, AX
MOV AX, 6000H
MOV ES, AX
MOV SI, OFFSET STRING1
MOV DI, OFFSET STRING2
MOV CX, 0010H
REPE CMPS
```

(iii) SCAS : 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 string is stored in CX. Whenever a match to the specified operand is found in the string, execution stops and zero flag is set otherwise reset.

Exe :-

```
MOV AX, 4000H
MOV ES, AX
MOV DI, OFFSET STRING
MOV CX, 0010H
MOV AX, WORD
REPNE SCAS
```

(iv) LODS : Load String Byte or String Word

This instruction loads the AL/AX register by the content of a string pointed by DS: SI register pair. The SI is modified automatically depending upon DF. If it is byte transfer, the SI is modified by one and if it is a word transfer (LODSW), the SI is modified by two.

Exe

```
MOV SI, OFFSET STRING
LODS STRING
```

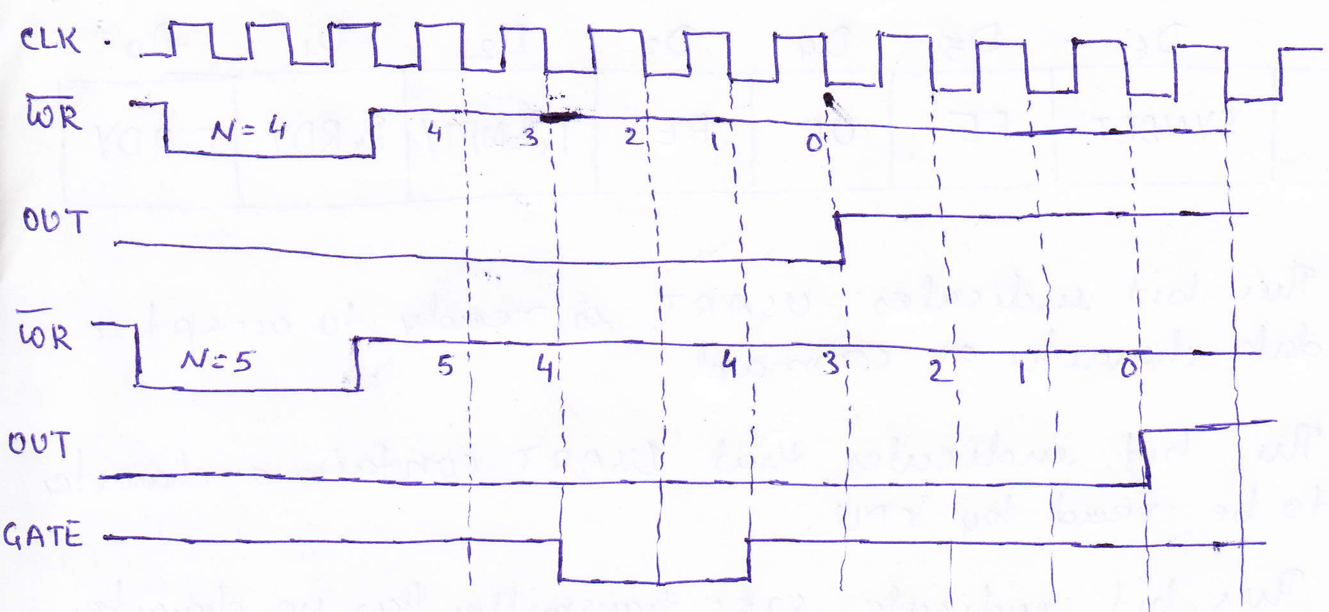
5(i) Difference between Programmed I/O and Interrupt driven I/O

Programmed I/O	Interrupt driven I/O
1) In programmed I/O, processer has to check each I/O device in sequence and in effect 'ask' each one if it needs communication with the processer. This checking is achieved by continuous polling cycle and hence processer can not execute other instruction.	1) External asynchronous input is used to tell the processer that I/O device needs its service and hence processer does not have to check whether I/O device needs its service or not.
2) During polling processer is busy and therefore, have serious and decremental effect on system throughput.	2) In interrupt driven I/O, the processer is allowed to execute its instruction in sequence and only stop to service I/O device when it is told to do so by the device itself. This increases system throughput
3) It is implemented without interrupt hardware support.	3) It is implemented using interrupt hardware support.
4) It does not depend on interrupt status.	4) Interrupt must be enabled to process interrupt driven I/O
5) It does not need initialization of stack.	5) It needs initialization of stack.
6) System throughput decreases as number of I/O devices connected in the system increases.	6) System throughput does not depend on number of I/O devices connected in the system.

5(ii) Operating mode of 8253 :

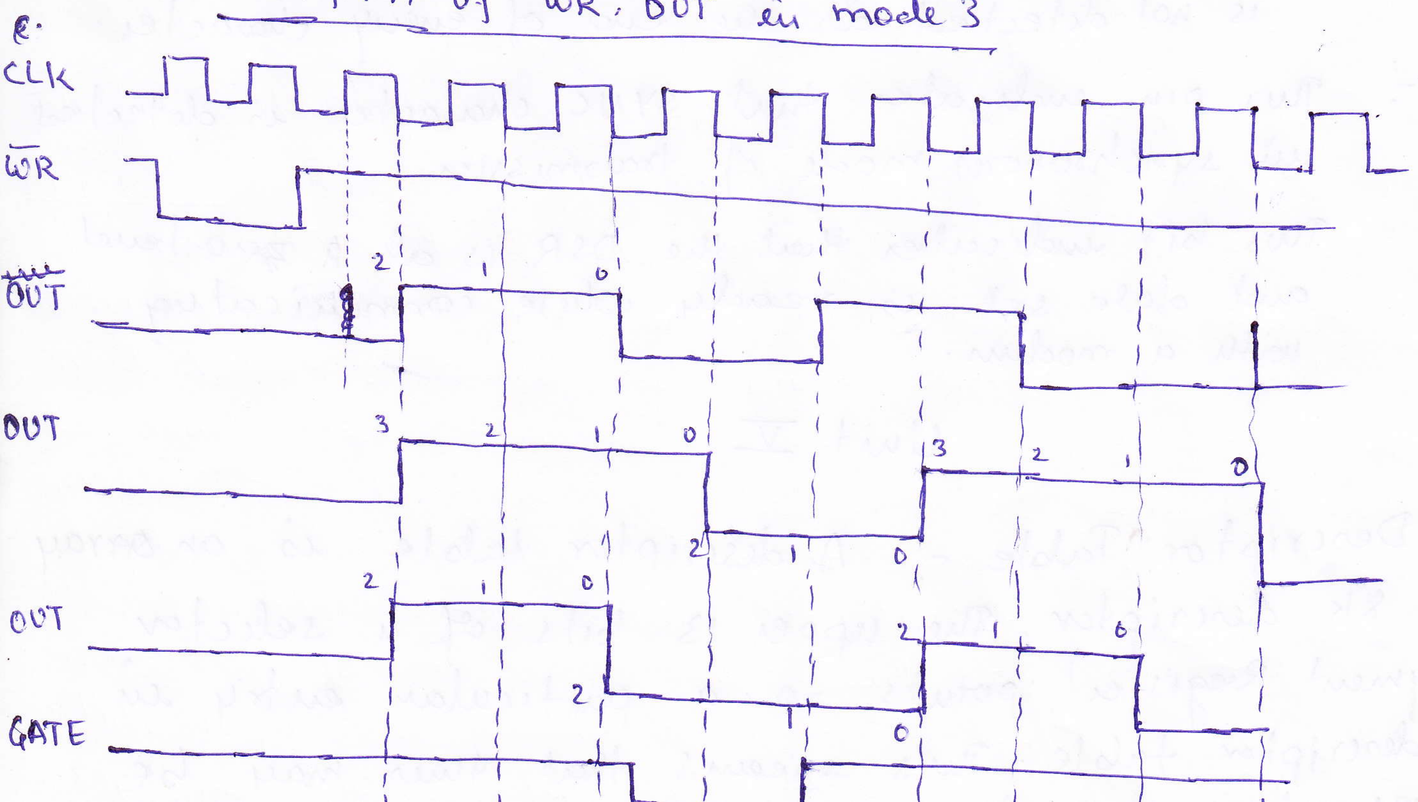
Mode 0 : This mode of operation is generally called as interrupt on terminal count. In this mode, the output is initially low after the mode is set. The output remains low even after the count value is loaded in counter. The counter starts decrementing the count value after the falling edge of the clock, if the GATE input is high. When the terminal count is reached, the output goes high and remain high till the selected control word register or the count register is reloaded with new mode of operation or a new count, The GATE signal is active high and should be high for normal counting. When GATE goes low counting is terminated and current count is

Wave form of $\overline{WR}$, OUT and GATE in mode 0



<u>Mode 3</u> : In this mode, the 8253 can be used as a square wave rate generator. In terms of operation ~~this mode~~, when the count N loaded is even, then for half of the count the output remain high and for remaining half it remains low. If the count is odd, the first clock pulse decrements it by 1 resulting in an even count value (holding the output high). Then the output remains high for half of the new count and goes low for the remaining half. If the GATE goes high, the counter starts counting down from the initial value; the counting is inhibited when GATE becomes low.

<u>Wave form of $\overline{WR}$, $\overline{OUT}$ in mode 3</u>



5(iii) Status Register format of USART 8251

D7	D6	D5	D4	D3	D2	D1	D0
DSR	SYNDET	FE	OE	PE	TxEMPTY	RxRDY	TxRDY

TxRDY :- This bit indicates, USART is ready to accept a data character or command

RxRDY :- This bit indicates that USART contain a character to be read by CPU.

TxEMPTY :- This bit indicate 8251 transmitter has no character to transmit.

PE (Parity Error) :- If set, indicates that at receiver end parity of character does not match with pre-defined parity, parity error occurs.

OE (Overrun Error) :- This flag is set when the CPU does not read a character before the next one becomes available

FE (Framing Error) :- This flag is set when a valid stop bit is not detected at the end of every character.

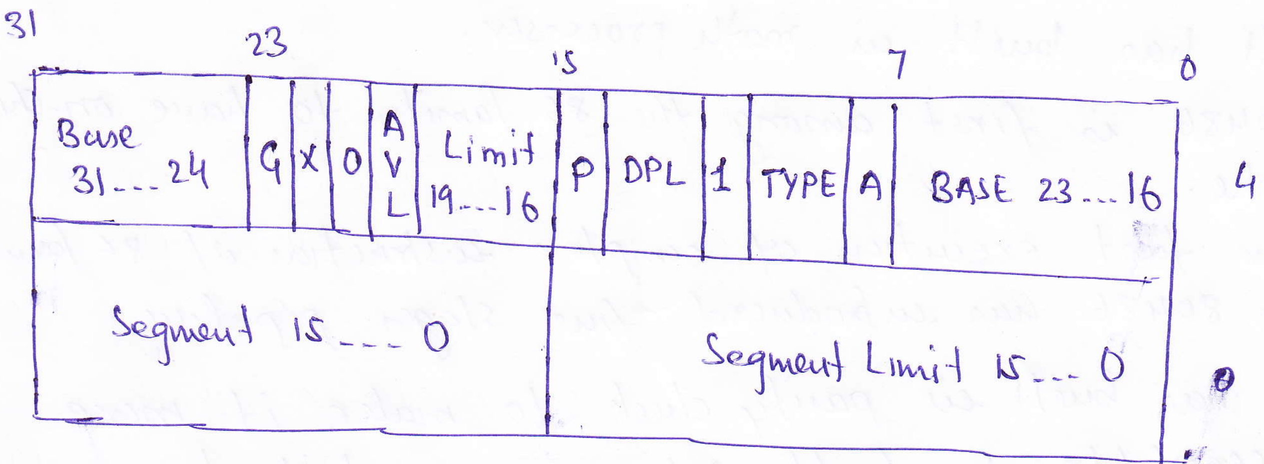
SYNDET :- This pin indicates that SYNC character is detected in synchronous mode of transmission.

DSR :- This bit indicates that the DSR is at a zero level and data set is ready while communicating with a modem.

Unit V

6(i) Descriptor Table :- A descriptor table is an array of 8K descriptor. The upper 13 bits of a selector (segment Register) points to a particular entry in a descriptor table. This means that there may be

A Gobal Descriptor Table (GDT) contain global descriptor common to all the taste. A Local Descriptor Table (LDT) contain descriptor specific to a particular task.



6(ii) Signal of 80386

(a) $BE_0\#$ - $BE_3\#$: The 32-bit databus supported by 80386 and the memory system of 80386 can be viewed as a 4-byte wide memory access mechanism. The four byte enable line $BE_0\#$ to $BE_3\#$ may be used for enabling these four banks. Using these four enable signal lines, the CPU may transfer 1byte/2bytes/3bytes or 4 bytes of data simultaneously.

(b) BS_{16} : The bus size 16 input pin allows the interfacing of 16-bit devices with the 32 bit wide 80386 data bus. Successive 16 bit bus cycles may be executed to read 32 bit data from a peripheral.

(c) ADS # : The address status output pin indicates that the address bus and bus cycle definition pin (W/R #, D/C #, M/IO #, $BE_0\#$ - $BE_3\#$) are carrying valid signals.

(d) NA # : The next address input pin, if activated allow address pipelining during 80836 bus cycle.

6(iii) Salient feature of 80486

(i) 80486 is 32 bit processor.

(ii) It has built-in math processor.

(iii) 80486 is first among the 86 family to have on-chip cache.

(iv) For fast execution of complex instruction of 86 family the 80486 has introduced five stages pipeline.

(v) It has built in parity check to make it more susceptible to fault-tolerance architecture.

(vi) 80486 has three mode of operation: Real mode, protected mode and virtual 8086 mode

(vii) The MMU provides four level of protection for isolating and protecting application and the operating system from each other.