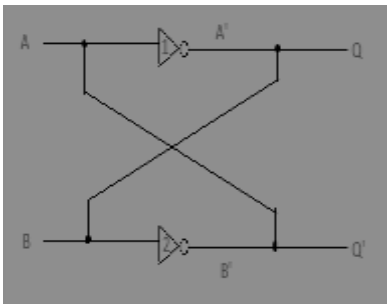
WINTER – 19EXAMINATION

Subject Name: Digital Techniques and Microprocessor **Model Answer** Subject Code: 22323

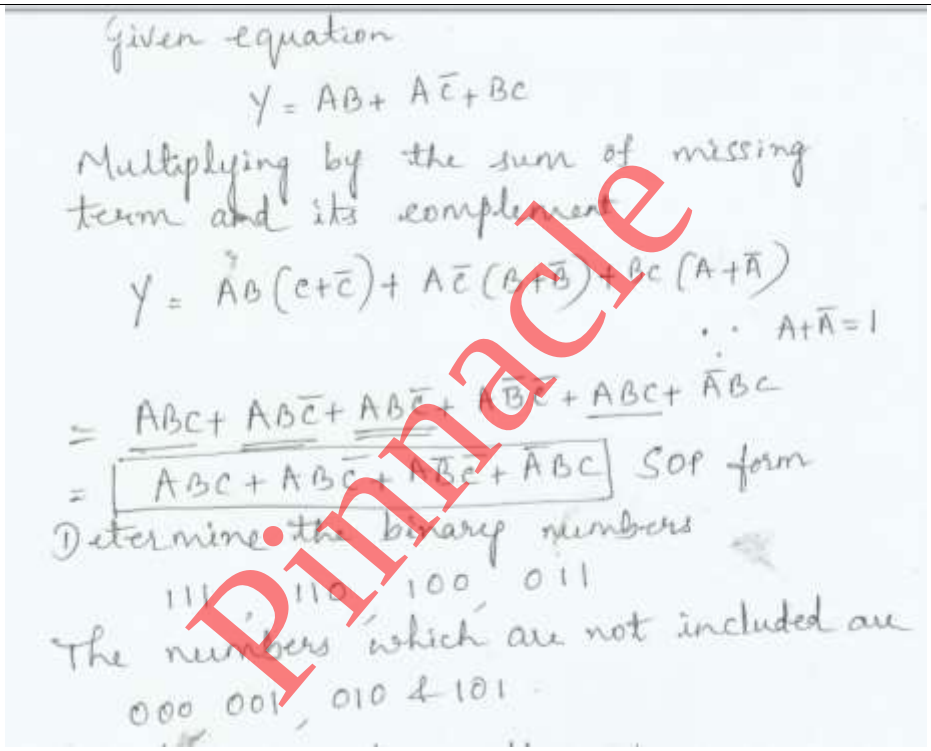
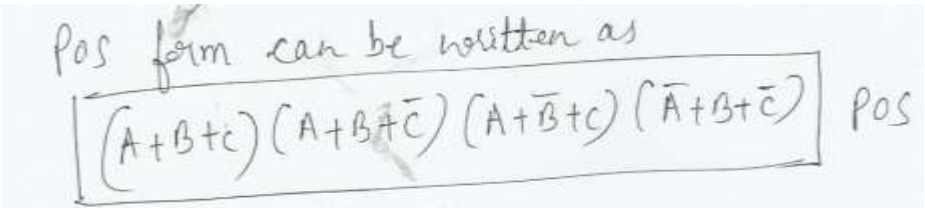
Important Instructions to examiners:

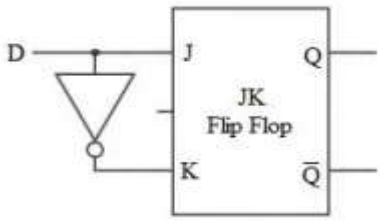
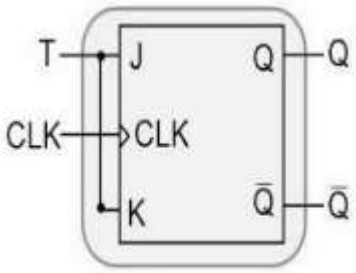
1) The answers should be examined by key words and not as word-to-word as given in themodel answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may tryto assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given moreImportance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in thefigure. The figures drawn by candidate and model answer may vary. The examiner may give credit for anyequivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constantvalues may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

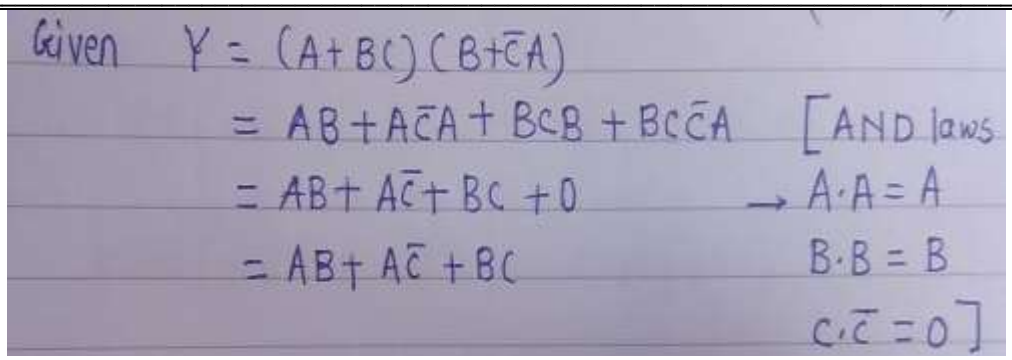
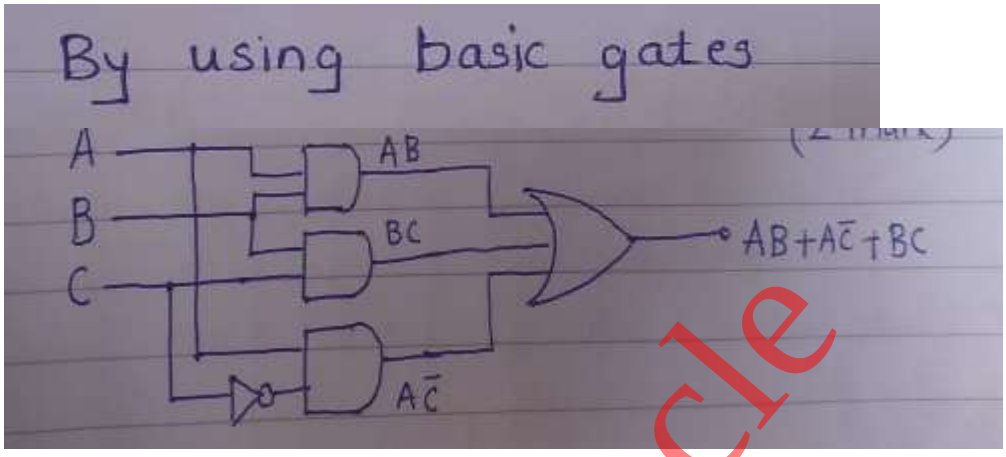
Q. No.	Sub Q. N.	Answer	Marking Scheme
Q.1		**Attempt any FIVE of the following:**	**10-Total Marks**
	a)	**List one application of each of following** **(i) Gray code** **(ii) ASCII code**	**2M**
	Ans:	(i) Gray codes are used for error correction in digital communication system. (ii)ASCII codes are used for identifying characters and numerals in a keyboard.	**1M Each**
	b)	**State the principle of multiplexer and mention its two types.**	**2M**
	Ans:	**Principle of multiplexer** Multiplexer is a circuit with many inputs and only one output. By applying control signals on select lines we can direct any input to the output. Types 4:1,16:1 etc	**1M** **1M**
	c)	**Draw the circuit of one bit memory cell.**	**2M**
	Ans:	Circuit : 	**2M**
	d)	**List features of 8086 microprpcessor.(Any four)**	**2M**

	Ans:	**(Note: Consider any 4)** Features of 8086 microprpcessor 1)It requires +5v power supply. 2)It has 20 bit address bus,can acceess 2^{20} =1MB memory location. 3)16 bit data bus. 4)It is a 16 bit processor having 16 bit ALU,16 bit registers. 5)It has instruction queue which is capable of storing 6 instruction bytes from the memory for faster processing. 6)It has pipelining,fetch and execute stagefor improving performance. 7)It has 256 vectored interrupts. 8)Clock range is 5-10 MHz.	**2M**
	e)	**Convert the following numbers into Hexadecimal number.** **(i) $(10110111)_2 = (?)_{16}$** **(ii) $(567)_8 = (?)_{16}$**	**2M**
	Ans:	(i) $(10110111)_2 = (B7)_{16}$ (ii) $(567)8 = (177)_{16}$	**2M**
	f)	**State four characteristics of RISC processor.**	**2M**
	Ans:	1)Reduced instruction set. 2)Simple addressing mode. 3)RISC processor consumes less power and has high performance. 4)Instruction is of uniform fixed length. 5)Large number of registers.	**2M**
	g)	**Give example of any two types of addressing mode of 8086**	**2M**
	Ans:	1)Direct addressing mode Eg:MOV CL,[1234] 2)Immediate addressing mode Eg:MOV AX,0005H 3)Register addressing mode Eg: MOV AX,BX 4)Base indexed addressing mode Eg:MOV CL,[BX+SI]	**Any two 1M each**

Q.2		**Attempt any THREE of the following:**	**12-Total Marks**
	a)	**Perform the following subtaction using 1's compliment and 2's compliment $(1010\ 0101)_2 - (1110\ 1110)_2$.**	**4M**
	Ans:	Subtaction using 1's compliment $(1010\ 0101)_2 - (1110\ 1110)_2$. Find 1's complement of the subtrahend 00010001 Add minuend 00010001 + 10100101	

		10110110 Since carry is 0,the resullt is –ve and in 1's complement form. The answer is -01001001 Subtaction using 2's compliment Find 1's complement of the subtrahend 00010001+1=00010010+ Add minuend 10100101 10110111 Since there is no carry,answer is –ve and is in its 2's complement form. The answer is -01001001	
	b)	**Simplify the given equation into standard SOP form Y = AB +A$\bar{C}$+ BC and represent the same equation in standard POS form.**	**4M**
	Ans:	  	**2M** **2M**
	c)	**Differentiate between D FF and T FF.**	**4M**

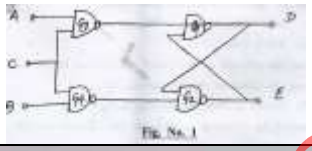
	Ans:		
		Input is transferred after a delay	When T=1, output toggles
		Used in shift registers	Used in counters, frequency dividers
		D Qn+1 0 0 **1 1**	T Qn+1 0 Qn **1 $\overline{Qn}$**
	d)	**Describe the characteristics of digital IC's (Any four).**	**4M**
	Ans:	Characteristics of digital IC's are 1)Fan out:It is the number of loads that the output of the gate can drive. 2)Power dissipation:Power consumed by the gate when fully driven by all its inputs. 3)Propagation delay:Time for the signal to propagate from input to output.\ 4)Noise margin:The maximum noise voltage added to an input signal that does not cause undesirable change in output. Fan in:It is the number of inputs connected to the gate without any degradation in the voltage level. Operating Temperature:It is the range of temperature in which the performance of IC is effective. Figure of merit:It is the product of speed and power.	**1M each**
Q.3		**Attempt any THREE of the following:**	**12-Total Marks**
	a)	**Reduce the following Boolean expression using laws of Boolean algebra and realize using logic gates.** **$Y = (A + BC)(B + \bar{C}A)$**	**4M**

	Ans:	 	2M 2M
	b)	**Write an assembly language program to transfer block of 10 numbers from one memory location to another.** **(Assume suitable data.)**	4M

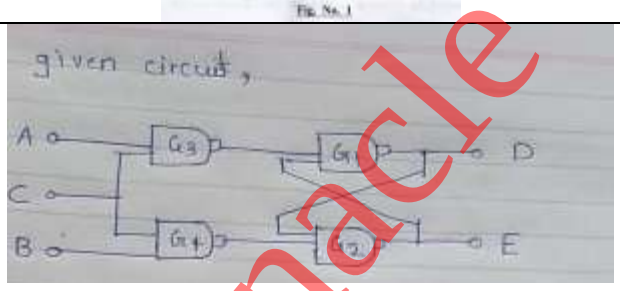
	Ans:	.model small
		.data
		src_arr dw 1,2,3,4,5,6,7,8,9,10
		dst_arr dw 10(dup) ;empty array
		.code
		mov ax,@data ;initialize data
		mov ds, ax
		mov cx,10 ; initialize counter
		mov si,offset src_arr ;initialize memory pointer for source
		mov di,offset dst_arr ;initialize memory pointer for destination
		up:
		mov ax,[si] ;read number from source array
			4M

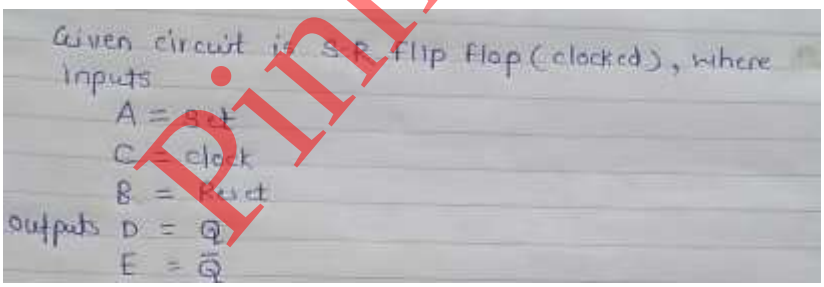
```
mov  [di] , ax          ;write number to destination array
add si,2                ;increment source memory pointer
add di,2                ;increment destination memory pointer
loop up                 ;check word counter for zero ,if not zero then read up number from
array
ends
end
```

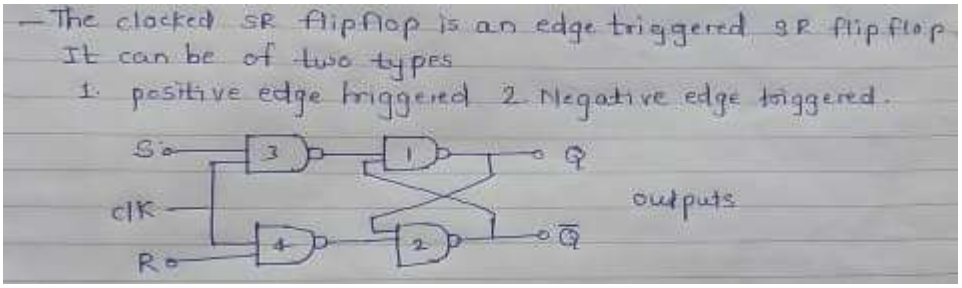
c) **For the given circuit, identify the inputs and outputs. Name the circuit and draw its truth table. Refer Fig .No. 1.** **4M**

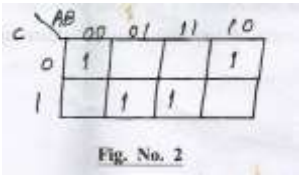
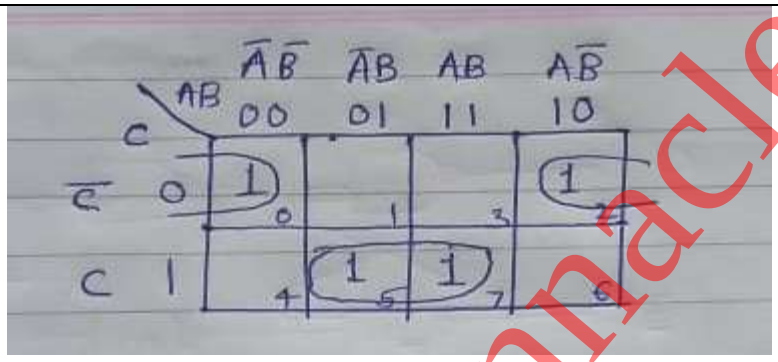
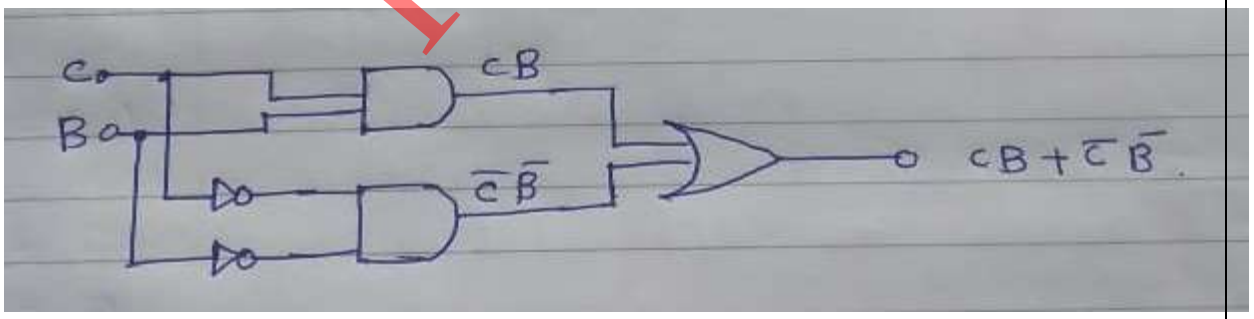


Ans: **4M**





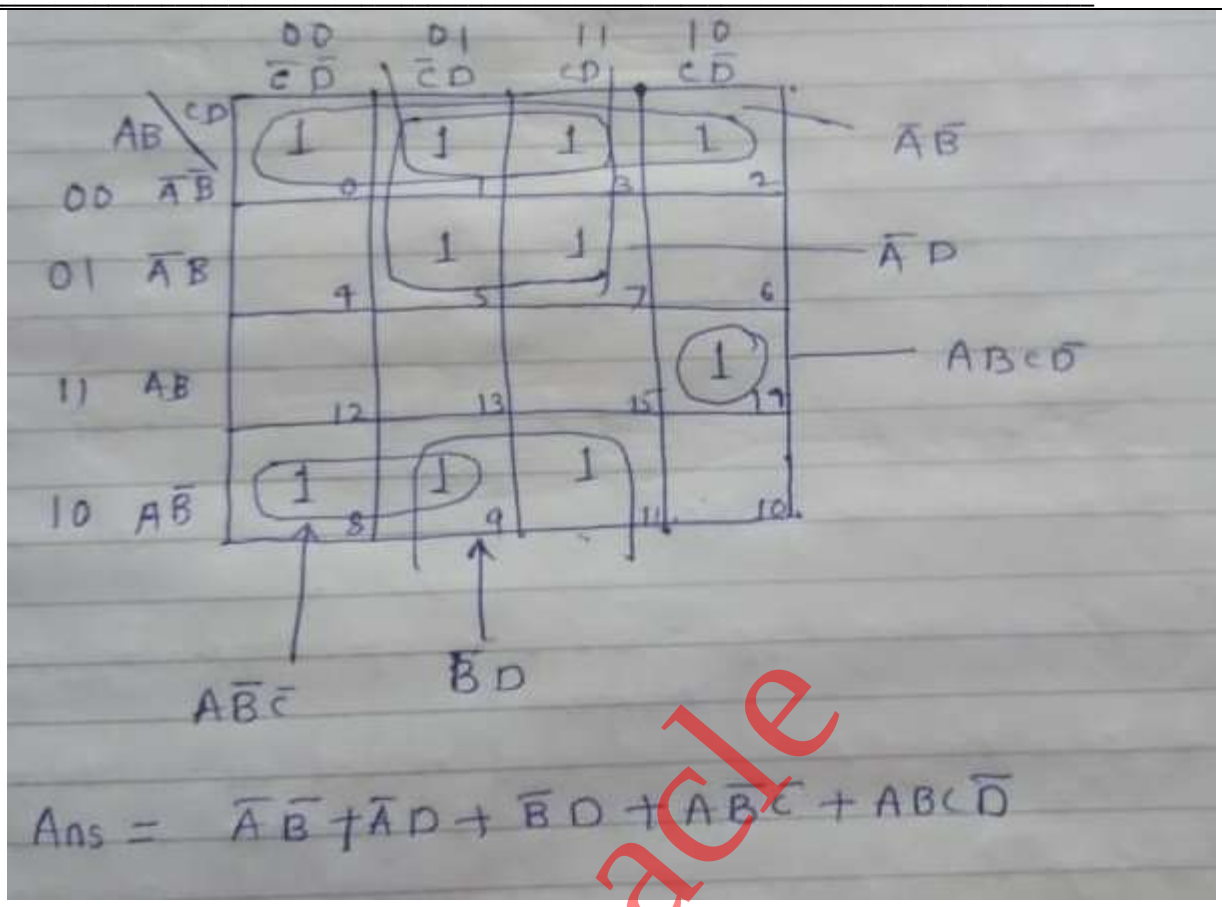


		Truth table Inputs: CLK, S, R; outputs: Q_{n+1}, $\bar{Q}_{n+1}$; Remark ↑ 0 0 → Q_n $\bar{Q}_n$ No change ↑ 0 1 → 0 1 Reset ↑ 1 0 → 1 0 Set ↑ 1 1 → Race Race Avoid	
	d)	**Simplify the given K-map using standard form and realize the circuit using gates. Refer Fig. No. 2.**  Fig. No. 2	**4M**
	Ans:	 Expression $CB + \bar{C}\bar{B}$ 	**2M** **2M**
Q.4		**Attempt any THREE of the following :**	**12-Total Marks**
	a)	**Write an assembly language program to find the sum of series of ten numbers stored in memory.(Assume suitable data.)**	**4M**
	Ans:	Sum Of Series:	**4M**

```
.model small
.data
    array   db   offh, 11h, 22h, 33h, 44h, 55h, 66h, 77h, 88h,
                 99h..
    Sum_lsb db   0
    Sum_msb db   0
.code
    mov ax,@data          ; Initialize data segment
    mov ds, ax
    mov cx, 10            ; initialize byte counter
    mov si, offset array  ; initialize memory pointer,
                                from
up:
    mov al, [si]          ; Read byte^ memory
    add sum_lsb, al       ; add with sum
    jnc next              ; if sum > 8 bit
    inc sum_msb           ; increment msb counter
next:
    inc si                ; Increment memory pointer
    loop up               ; decrement byte counter
                          ; if byte counter=0 then exit
                            else read next number

    ends
end
```

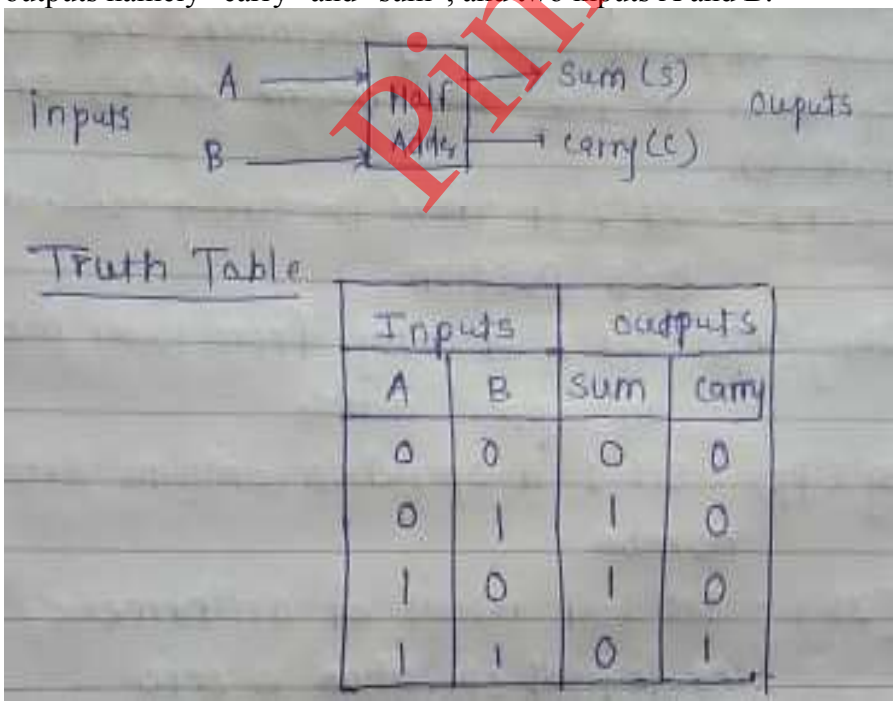
	b)	**Minimize the four variable logic function using K- map.** **F(A,B,C,D) Σm (0, 1, 2, 3, 5, 7, 8, 9, 11, 14)**	**4M**
	Ans:	f(A,B,C,D) Σm(0,1,2,3,5,7,8,9,11,14)	**4M**

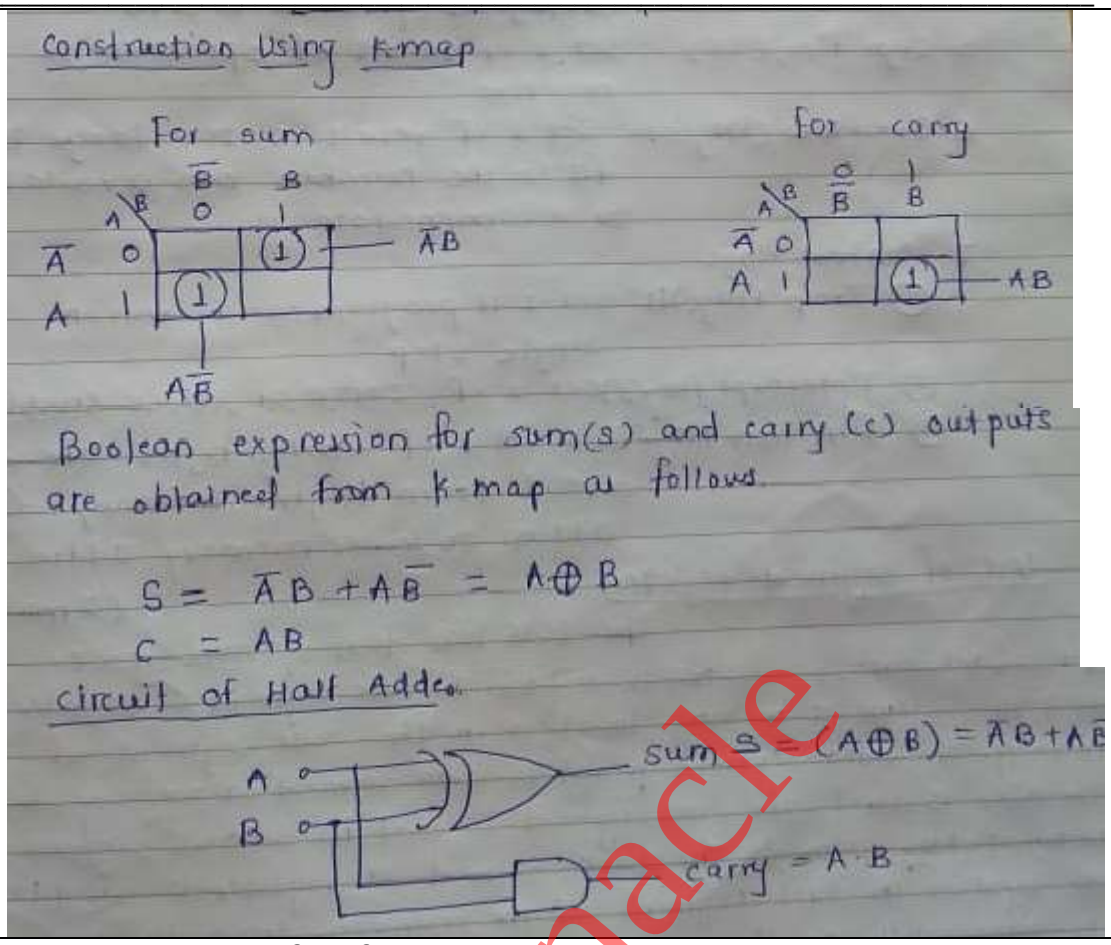


	c)	**Differentiate between sequential and combinational logic circuits. (Any four points)**	**4M**
	Ans:		**4M**

Sr.no	parameter	Sequential circuit	Combinational circuit
1	Output depends on	present inputs and past inputs/outputs	Input present at that instant of time
2.	Memory	Necessary	Not necessary
3.	clock input	Necessary	Not necessary
4	Example	Flip flops, shift registers, counters	Adder, substractors, code converters

	d)	**Describe the use of flag register and segment registers in 8086.**	**4M**

	Ans:	• Use of Flag Register: Microprocessor 8086 has 16 bit flag register among which 9 bits are active. The purpose of flag register is to indicate the status of the processor Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0). 1. Carry Flag (CF): Set 1 if there is carry out of MSB position. 2. Auxiliary Flag (AF): Set 1 if carry from lower nibble to upper nibble. 3. Parity Flag (PF): Set 1 if operation contains even number. 4. Zero Flag (ZF): Set 1 if result of arithmetic or logical operation is zero. 5. Sign Flag (SF): Set 1 if result of operation is negative. 6. Overflow Flag (OF): Set 1 if result is too large to fit in the numbers bits available to accommodate it. 7. Control Flags: (i) Trap Flag (TF): Set 1 if program can be run in single step. (ii) Interrupt Flag (IF): Set 1 if INTR of 8086 is enabled. (iii) Direction Flag (DF): Set 1 if string bytes are write or read from higher memory address to lower memory address. • Use of Segment Register: The 8086 has four segment register of 16 bit each. i.e. CS,DS,SS and ES. The code segment CS register used to address a memory location in the code segment of memory. The data segment point to data segment of memory where the data is stored the extra segment ES used to address the segment is additional data segment. The Stack segment SS register is used to point location in stack segment of the memory, used to store data temporarily on the stack.	**2M** **2M**
	e)	**Describe the construction of half adder using K – map.**	**4M**
	Ans:	Half Adder using k-map: Half adder is a combinational logic circuit with two inputs and two output , circuit has two outputs namely “carry” and “sum”, and two inputs A and B.  	**2M**

			2M
Q.5		**Attempt any TWO of the following**	**12-Total Marks**
	(a)	**Write an assembly language program to find the factorial of a number using looping process.**	**6M**

	Ans:	DATA SEGMENT
		A DW0005H
		FACT_LSBDW?
		FACT_MSBDW?
		DATA ENDS
		CODE SEGMENT
		ASSUME DS:DATA,CS:CODE
		START:MOVAX,DATA
		MOV DS,AX
		CALL FACTORIAL
		MOVAH,4CH
		INT 21H
		FACTORIAL PROC
		MOV AX,A
		MOV BX,AX
		DEC BX
			6M

```
UP: MUL BX ; MULTIPLY AX*BX
MOV FACT_LSB,AX ; ANS DX:AX PAIR
MOV FACT_MSB,DX
DEC BX
CMP BX,0
JNZ UP
RET
FACTORIAL ENDP
```

OR

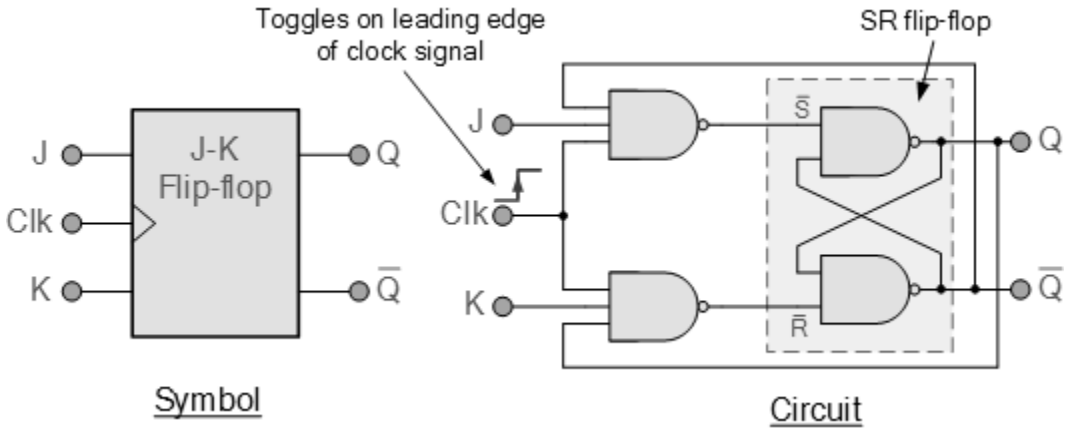
```
DATA SEGMENT
 NUM DB 05H
 RES DW ?
DATA ENDS
CODE SEGMENT
 ASSUME CS:CODE,DS:DATA
START:
MOVAX,DATA
MOV DS,AX
 CALL FAC
MOV RES,AX
MOVAH,4CH
 INT 21H
FAC PROC
MOVCL,NUM
 DEC CL
MOV AL,NUM
MOVAH,00H
MOV BL,CL
MOV BH,00H
L1: MUL BX
 DEC BX
 DEC CL
JNZ L1
 RET
FAC ENDP
CODE ENDS
END START
```

Correct Program with any other logic can be given marks.

	(b)	**Describe the principle of working of JK FF and draw its circuit diagram and truth table.**	**6M**
	Ans:	The **JK flip flop** is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level "1". Due to this additional clocked input, a JK flip-flop has four	

possible input combinations, "logic 1", "logic 0", "no change" and "toggle". The symbol for a JK flip flop is similar to that of an *SR Bistable Latch* as seen in the previous tutorial except for the addition of a clock input.

The Basic JK Flip-flop



2M

Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs. Then this equates to: J = S and K = R.

The two 2-input AND gates of the gated SR bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q. This cross coupling of the SR flip-flop allows the previously invalid condition of S = "1" and R = "1" state to be used to produce a "toggle action" as the two inputs are now interlocked.

If the circuit is now "SET" the J input is inhibited by the "0" status of Q through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q are always different we can use them to control the input. When both inputs J and K are equal to logic "1", the JK flip flop toggles as shown in the following truth table.

The Truth Table for the JK Function

CLK	J	K	Q_{n+1}	$\overline{Q}_{n+1}$	Description
1 , 0 , ↑	X	X	Q_n	$\overline{Q}_n$	No Change
↓	0	0	Q_n	$\overline{Q}_n$	No Change
↓	0	1	0	1	Reset Condition
↓	1	0	1	0	set Condition
↓	1	1	$\overline{Q}_n$	Q_n	Toggle condition

2M

Then the JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit.

Also when both the J and the K inputs are at logic level "1" at the same time, and the clock input is pulsed "HIGH", the circuit will "toggle" from its SET state to a RESET state, or visa-versa. These results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are "HIGH".

2M

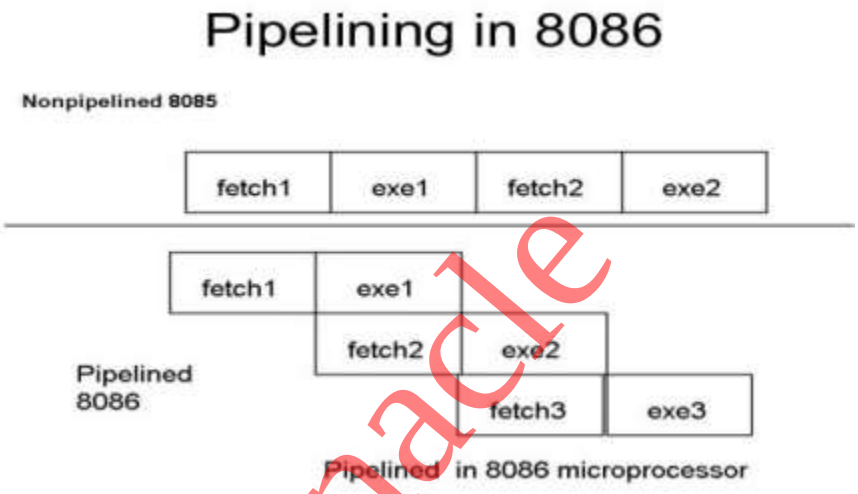
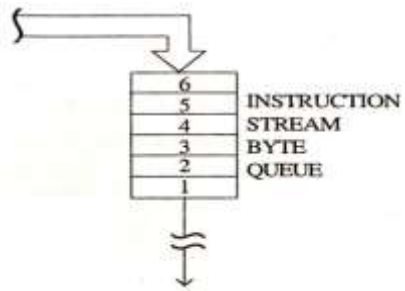
Although this circuit is an improvement on the clocked SR flip-flop it still suffers from timing problems called "race" if the output Q changes state before the timing pulse of the clock input has time to go "OFF". To avoid this the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC's the much improved **Master-Slave JK Flip-flop** was developed.

	(c)	**Differentiate between CISC and RISC and justify use of each of them in practice.**	**6M**
	Ans:	The architecture of the Central Processing Unit (CPU) operates the capacity to function from	**2M**

"Instruction Set Architecture" to where it was designed. The architectural design of the CPU is Reduced instruction set computing (RISC) and Complex instruction set computing (CISC). CISC has the capacity to perform multi-step operations or addressing modes within one instruction set. It is the CPU design where one instruction works several low-level acts. For instance, memory storage, loading from memory, and an arithmetic operation. Reduced instruction set computing is a Central Processing Unit design strategy based on the vision that basic instruction set gives a great performance when combined with a microprocessor architecture which has the capacity to perform the instructions by using some microprocessor cycles per instruction. The hardware part of the Intel is named as Complex Instruction Set Computer (CISC), and Apple hardware is Reduced Instruction Set Computer (RISC).

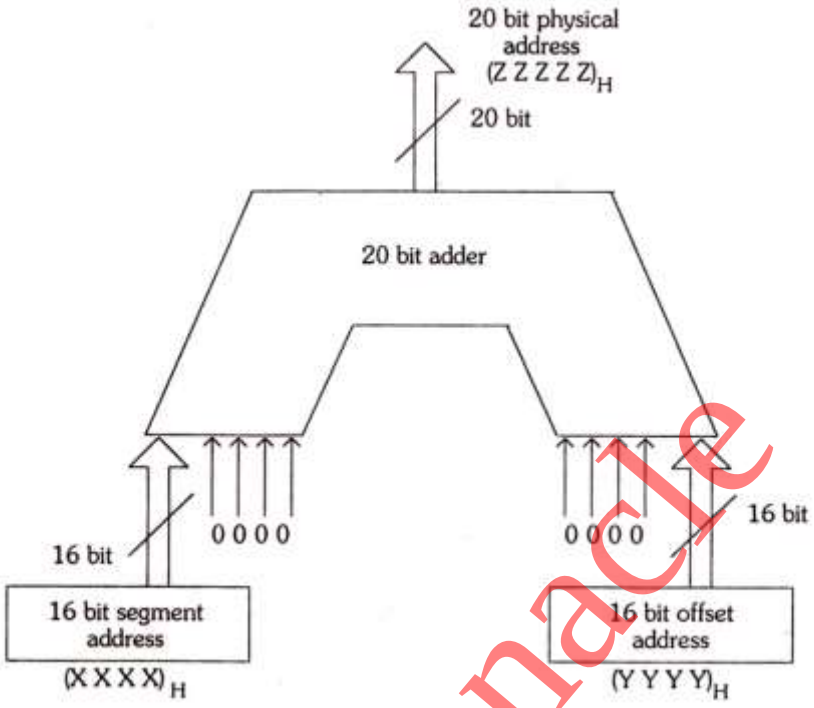
Sr. No.	CISC	RISC
1	A large number of instructions are present in the architecture.	Very fewer instructions are present. The number of instructions are generally less than 100.
2	Some instructions with long execution times. These include instructions that copy an entire block from one part of memory to another and others that copy multiple registers to and from memory.	No instruction with a long execution time due to very simple instruction set. Some early RISC machines did not even have an integer multiply instruction, requiring compilers to implement multiplication as a sequence of additions.
3	Variable-length encodings of the instructions.	Fixed-length encodings of the instructions are used.
4	Example: IA32 instruction size can range from 1 to 15 bytes.	Example: In IA32, generally all instructions are encoded as 4 bytes.
5	Multiple formats are supported for specifying operands. A memory operand specifier can have many different combinations of displacement, base and index registers.	Simple addressing formats are supported. Only base and displacement addressing is allowed.
6	CISC supports array.	RISC does not supports array.
7	Arithmetic and logical operations can be applied to both memory and register operands.	Arithmetic and logical operations only use register operands. Memory referencing is only allowed by load and store instructions, i.e. reading from memory into a register and writing from a register to memory respectively.
8	Implementation programs are hidden from machine level programs. The ISA provides a clean abstraction between programs and how they get executed.	Implementation programs exposed to machine level programs. Few RISC machines do not allow specific instruction sequences.
9	Condition codes are used.	No condition codes are used.
10	The stack is being used for procedure arguments and return addresses.	Registers are being used for procedure arguments and return addresses. Memory references can be avoided by some procedures.

4M

Q.6		Attempt any TWO of the following:	12Total Marks
	(a)	**Describe the concept of pipelining and process of physical address generation in 8086 microprocessor.**	6M
	Ans:	CONCEPT OF PIPELINING Fetching the next instruction while the current instruction executes is known as pipelining it means When first instruction is getting executed, second one's is decoded and third instruction code is fetched from memory. This process is known as pipelining. It improves speed of operation to great extent.  Pipelined in 8086 microprocessor To speed up program execution, the Bus Interface Unit(BIU) fetches as many as 6 instruction bytes ahead of time from the memory and these are held for execution unit in the (FIFO) group of registers called QUEUE. The BIU can fetch instruction bytes while EU is decoding or executing an instruction which does not require the use of buses. When the EU is ready for the next instruction, it simply reads the instruction from the QUEUE in the BIU. This is much faster than sending out addresses to system memory and waiting for the memory to send back the next instruction byte. The Queue is refilled when at least two bytes are empty as 8086 has a 16-bit data bus. In case of Branch instructions however, the instructions pre-fetched in the queue are of no use. Hence the QUEUE has to be dumped and new instructions are fetched from the destination addresses specified by the branch instructions.  	3M

Physical Address Generation:

The 8086 addresses a segmented memory. The complete physical address which is 20-bits long is generated using segment and offset registers each of the size 16-bit.The content of a segment register also called as segment address, and content of an offset register also called as offset address. To get total physical address, put the lower nibble 0H to segment address and add offset address. The figure shows formation of 20-bit physical address.



3M

The physical address is calculated as follows:

```
  X X X X 0
+   0 Y Y Y Y
-------------
  Z Z Z Z Z
```

Where $(X\ X\ X\ X)_H \rightarrow$ 16 bit segment address

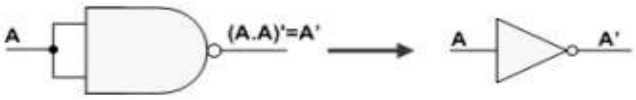
$(Y\ Y\ Y\ Y)_H \rightarrow$ 16 bit offset address

and $(Z\ Z\ Z\ Z\ Z)_H \rightarrow$ 20 bit offset address

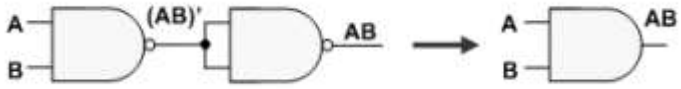
- A segment is a 64 K block, hence, there could be 16 non overlapping segments in 1 MB of memory.
- The size of any segment cannot be greater than 64KB.
- The size of any segment cannot be lesser than 16 Byte.

	(b)	**State the names of universal logic gates and design basic gates using universal gates.**	**6M**
	Ans:	**Universal logic gates :** NAND gate NOR gate NAND GATE AS AN UNIVERSAL GATES:	**1M** **NAND:2.5 M** **NOR:2.5M**

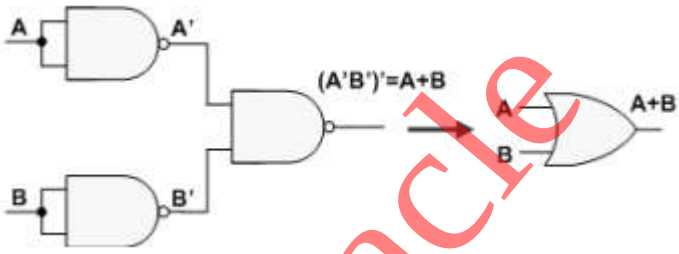
1. Implementing an Inverter Using only NAND Gate The figure shows two ways in which a NAND gate can be used as an inverter (NOT gate).All NAND input pins connect to the input signal A gives an output A'.



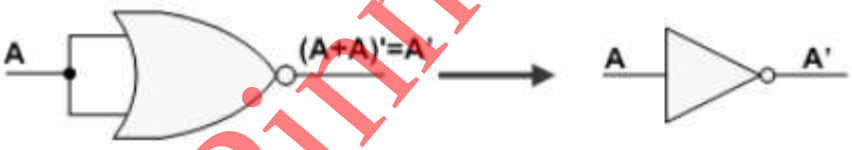
2. Implementing AND Using only NAND Gates An AND gate can be replaced by NAND gates as shown in the figure (The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter).



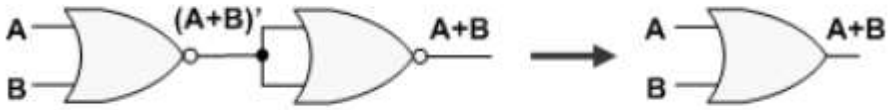
3. Implementing OR Using only NAND Gates An OR gate can be replaced by NAND gates as shown in the figure (The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate.
inverters)



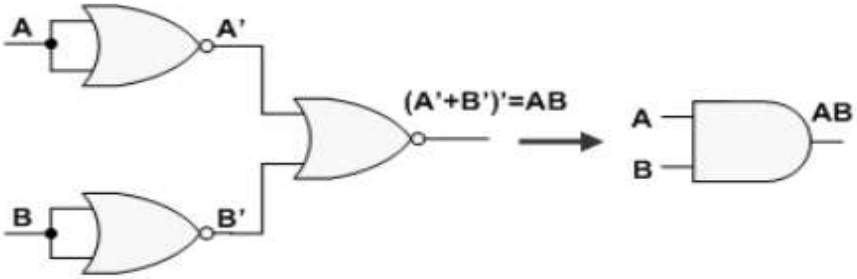
NOR GATE AS AN UNIVERSAL GATES:
1. All NOR input pins connect to the input signal A gives an output A'.



2. Implementing OR Using only NOR Gates An OR gate can be replaced by NOR gates as shown in the figure (The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter).



3. An AND gate can be replaced by NOR gates as shown in the figure (The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters)



	(c)	**Describe the use of shift and rotate instructions as well as string instructions with the**	**6M**

help of one relevant examples of each.

Ans:

SHIFT AND ROTATE INSTRUCTIONS

In the 8086 microprocessor, we have 16-bit registers to handle our data. Sometimes, the need to perform some necessary shift and rotate operations on our data may occur according to the given condition and requirement. So, for that purpose, we have various Shift and Rotate instructions present in the 8086 microprocessor.

1) SHR : Shift Right

The SHR instruction is an abbreviation for 'Shift Right'. This instruction simply shifts the mentioned bits in the register to the right side one by one by inserting the same number (bits that are being shifted) of zeroes from the left end. The rightmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHR Register, Bits to be shifted
Example: SHRAX, 2

Working:

2) SAR : Shift Arithmetic Right

The SAR instruction stands for 'Shift Arithmetic Right'. This instruction shifts the mentioned bits in the register to the right side one by one, but instead of inserting the zeroes from the left end, the MSB is restored. The rightmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SAR Register, Bits to be shifted
Example: SAR BX, 5

Working:

2) SHL : Shift Left

The SHL instruction is an abbreviation for 'Shift Left'. This instruction simply shifts the mentioned bits in the register to the left side one by one by inserting the same number (bits that are being shifted) of zeroes from the right end. The leftmost bit that is being shifted is stored in the Carry Flag (CF).

Syntax: SHL Register, Bits to be shifted
Example: SHLAX, 2

Working:

3) SAL : Shift Arithmetic Left

The SAL instruction is an abbreviation for 'Shift Arithmetic Left'. This instruction is the same as SHL.

INSTRUCTION:1M ,EXAMPLE:1M EACH

Syntax: SAL Register, Bits to be shifted
Example: SAL CL, 2

Working:

5) ROL : Rotate Left

The ROL instruction is an abbreviation for 'Rotate Left'. This instruction rotates the mentioned bits in the register to the left side one by one such that leftmost bit that is being rotated is again stored as the rightmost bit in the register, and it is also stored in the Carry Flag (CF).

Syntax: ROL Register, Bits to be shifted
Example: ROL AH, 4

Working:

6) ROR : Rotate Right

The ROR instruction stands for 'Rotate Right'. This instruction rotates the mentioned bits in the register to the right side one by one such that rightmost bit that is being rotated is again stored as the MSB in the register, and it is also stored in the Carry Flag (CF).

Syntax: ROR Register, Bits to be shifted
Example: ROR AH, 4

Working:

7) RCL : Rotate Carry Left

This instruction rotates the mentioned bits in the register to the left side one by one such that leftmost bit that is being rotated it is stored in the Carry Flag (CF), and the bit in the CF moved as the LSB in the register.

Syntax: RCL Register, Bits to be shifted
Example: RCL CH, 1

Working:

8) RCR : Rotate Carry Right

This instruction rotates the mentioned bits in the register to the right side such that rightmost bit that is being rotated it is stored in the Carry Flag (CF), and the bit in the CF moved as the MSB in the register.

Syntax: RCR Register, Bits to be shifted
Example: RCRBH, 6

Working:

STRING INSTRUCTIONS

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

1. MOVS/MOVSB/MOVSW Instruction :

This instruction copies a byte or word from a location in the data segment to a location in the extra segment. The offset of the source byte or word in the data segment must be in the SI register. The offset of the destination in the extra segment must be contained in the DI register. For multiple byte or multiple word moves the number of elements to be moved is put in the CX register so that it can function as a counter. After the byte or word is moved SI and DI are automatically adjusted to point to the next source and the next destination. If the direction flag is 0, then SI and DI will be incremented by 1 after a byte move and they will incremented by 2 after a word move. If the DF is a 1, then SI and DI will be decremented by 1 after a byte move and they will be decremented by 2 after a word move. MOVS affects no flags.

Examples :

```
CLD                     ;  Clear Direction Flag to autoincrement SI
                        ;  and DI
MOV  AX,  0000H
MOV  DS,  AX            ;  Initialize data segment register to 0
MOV  ES,  AX            ;  Initialize extra segment register to 0
MOV  SI,  2000H         ;  Load offset of start of source string
                        ;  into SI
MOV  DI,  2400H         ;  Load offset of start of destination into DI
MOV  CX,  04H           ;  Load length of string in CX as counter
REP  MOVSB              ;  Decrement CX and MOVSB until CX will be 0.
```

2. CMPS/CMPSB/CMPSW Instruction :

A 8086 String Instructions is a series of the same type of data items in sequential memory locations. The CMPS instruction can be used to compare a byte in one string with a byte in another string or to compare a word in one string with a word in another string. SI is used to hold the offset of a byte or word in the source string and DI is used to hold the offset of a byte or a word in the other string. The comparison is done by subtracting the byte or word pointed to by DI from the byte or word pointed to by SI. The AF, CF, OF, PF, SF, and ZF flags are affected by the comparison, but neither operand is affected.

Examples :

```
MOV  SI,  OFFSET  F_STRING    ;  Point SI at source string, Point DI
MOV  DI,  OFFSET  S_STRING    ;  at destination string
CLD

CMPS  F_STRING,  S_STRING     ;  DF cleared so SI and DI will
                              ;  autoincrement after compare
                              ;  The assembler uses names to determin
                              ;  whether strings were declared as typ
                              ;  byte or as type word.
MOV  CX,  100                 ;  Put number of string elements in CX,
                              ;  Point SI at source of string and DI
                              ;  at destination of string
MOV  SI,  OFFSET  F_STRING
MOV  DI,  OFFSET  S_STRING
STD

REPE  CMPSB                   ;  DF set so SI and DI will
                              ;  autodecrement after compare
                              ;  Repeat the comparison of string byte
                              ;  until end of string or until compare
                              ;  bytes are not equal.
```

After the comparison SI and DI will be automatically incremented or decremented according to direction flag to point to the next element in the two strings (if DF = 0, SI and DI ↑) CX functions as a counter which is decremented after each comparison. This will go on until CX = 0

3. SCAS/SCASB/SCASW Instruction :

SCAS compares a string byte with a byte in AL or a string word with word in AX. The instruction affects the flags, but it does not change either the operand in AL (AX) or the operand in the 8086 String Instructions. The string to be 'scanned must be in the extra segment and DI must contain the offset of the byte or the word to be compared.

After the comparison DI will be automatically incremented or decremented according to direction flag, to point to the next element in the two strings (if DF = 0, SI and DI ↑) CX functions as a counter which is decremented after each comparison. This will go on until CX = 0. SCAS affects the AF, CF, OF, PF, SF and ZF flags.

Examples :

```
                                ; Scan a text string of 80
                                ; characters for a carriage
                                ; return
MOV  AL,  0DH                   ; Byte to be scanned for into AL
MOV  DI,  OFFSET  TEXT_STRING   ; Offset of string to DI
MOV  CX,  80                    ; CX used as element counter
CLD                             ; Clear DF, so DI
                                ; autoincrements
REPNE  SCAS  TEXT_STRING        ; Compare byte in string with
                                ; byte in AL.
```

SCASB says compare 8086 String Instructions as bytes and SCASW says compare strings as words.

4. LODS/LODSB/LODSW Instruction :

This instruction copies a byte from a string location pointed to by SI to AL, or a word from a string location pointed to by SI to AX. LODS does not affect any flags. LODSB copies byte and LODSW copies a word.

Examples :

```
CLD                          ; Clear direction flag so SI
                             ; is autoincremented
MOV SI, OFFSET S_STRING      ; Point SI at string
LODS  S_STRING.
```

5. STOS/STOSB/STOSW Instruction :

The STOS instruction copies a byte from AL or a word from AX to a memory location in the extra segment. DI is used to hold the offset of the memory location in the extra segment. After the copy, DI is automatically incremented or decremented to point to the next string element in memory. If the direction flag, DF, is cleared, then DI will automatically be incremented by one for a byte string or incremented by two for a word 8086 String Instructions. If the direction flag is set, DI will be automatically decremented by ono for a byte string or decremented by two for a word string. STOS does not affect any flags. STOSB copies byte and STOSW copies a word.

Examples :

```
MOV  DI,  OFFSET  D_STRING   ; Point DI at destination string
STOS  D_STRING               ; Assembler uses string name to
                             ; determine whether string is of
                             ; type byte or type word. If byte
                             ; string, then string byte replaced
                             ; with contents of AL. If word
                             ; string, then string word replaced
                             ; with contents of AX.
MOV  DI,  OFFSET  D_STRING   ; Point DI at destination string
STOSB                        ; "B" added to STOS mnemonic
                             ; directly tells assembler to
                             ; replace byte in string with
                             ; from AL. STOSW would tell assembler
                             ; directly to replace a word in
                             ; the string with a word from AX.
```