Digital Techniques and Microprocessors (313305)

Model Question Paper — Set C · Model Answers

Q.1

a) Gray code and BCD. 2 M

i) $(10110110)_2$ → Gray

Binary → Gray: MSB is copied as it is; every next Gray bit = XOR of the present and previous binary bits.

Binary	1	0	1	1	0	1	1	0
XOR step	(copy)	1⊕0	0⊕1	1⊕1	1⊕0	0⊕1	1⊕1	1⊕0
Gray	1	1	1	0	1	1	0	1

$\therefore (10110110)_2 = (11101101)_{Gray}$

ii) $(59)_{10}$ → BCD (each decimal digit as 4-bit binary)

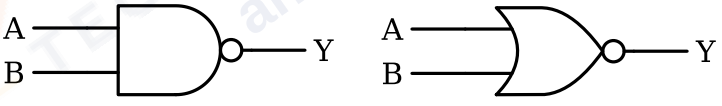
Decimal digit	5	9
BCD (8421)	0101	1001

$\therefore (59)_{10} = (0101\ 1001)_{BCD}$

(1 M each)

b) Universal gates. 2 M

NAND and NOR gates are called **universal gates** because any Boolean function and every basic gate (NOT, AND, OR, EX-OR …) can be implemented using only NAND gates or only NOR gates. Hence a complete digital circuit can be built from a single gate type (cheaper, easy to manufacture).



NAND: Y = (AB)' *NOR: Y = (A + B)'*

(Reason 1 M, symbols 1 M)

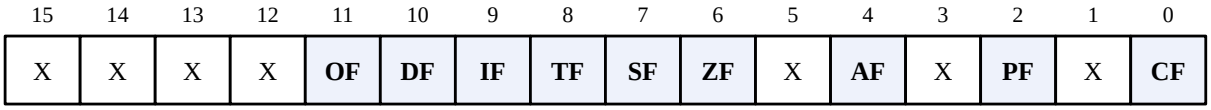
c) Applications of flip-flop and counter. 2 M

Flip-flop: i) 1-bit memory / storage element (registers, SRAM) ii) shift registers, frequency division by 2, switch debouncing.

Counter: i) counting events / pulses, digital clocks and watches ii) frequency division, timers, generating addresses (program counter).

(½ M each)

d) Flag register format. 2 M



Status flags (6): CF, PF, AF, ZF, SF, OF

Control flags (3): TF, IF, DF

Fig.: Flag register of 8086 (X = undefined / reserved)

CF – carry, PF – parity, AF – auxiliary carry, ZF – zero, SF – sign, TF – trap, IF – interrupt enable, DF – direction, OF – overflow.

(Format 2 M)

e) Addressing modes of 8086. 2 M

1. Immediate addressing — `MOV AL, 25H`
2. Register addressing — `MOV AX, BX`
3. Direct addressing — `MOV AL, [3000H]`
4. Register indirect addressing — `MOV AL, [BX]`
5. Based addressing — `MOV AX, [BX + 04H]`
6. Indexed addressing — `MOV AL, [SI + 02H]`
7. Based indexed addressing — `MOV AL, [BX + SI]`
8. Based indexed with displacement — `MOV AL, [BX + DI + 06H]`
9. Others: string, implied (e.g. `CLC`), I/O port (`IN AL, 80H`), relative (jumps).

(Any four, ½ M each)

f) DD and EQU directives. 2 M

DD (Define Double word): reserves 4-byte (32-bit) locations and initialises them.
Example: `BIG DD 12345678H PROD DD ?`

EQU (Equate): gives a name to a constant value or expression; no memory is reserved — the assembler replaces the name by its value.
Example: `COUNT EQU 0AH` → `MOV CX, COUNT`

(1 M each)

g) ALP — addition of two 8-bit numbers. 2 M

```
DATA SEGMENT
    NUM1  DB 4BH
    NUM2  DB 39H
    SUM   DB ?
    CARRY DB 00H
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
       MOV DS, AX
       MOV AL, NUM1
```

```
        ADD AL, NUM2        ; AL = NUM1 + NUM2
        MOV SUM, AL
        JNC DONE
        INC CARRY
DONE:   MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

Result (checked by running the program on an 8086 simulator):
4BH + 39H → **SUM = 84H, CARRY = 00H**

(Logic 1 M, program 1 M)

Q.2

a) De Morgan's theorems; reduction by Boolean laws. 4 M

De Morgan's theorems: i) $\overline{A + B} = \overline{A} \cdot \overline{B}$ ii) $\overline{A \cdot B} = \overline{A} + \overline{B}$

i) Y = AB + A$\overline{B}$ + $\overline{A}$B

```
Y = AB + AB' + A'B
  = A(B + B') + A'B          (distributive law)
  = A·1 + A'B = A + A'B      (B + B' = 1)
  = (A + A')(A + B)          (A + BC = (A + B)(A + C))
  = A + B
```

ii) Y = AB + A(B + C) + B(B + C)

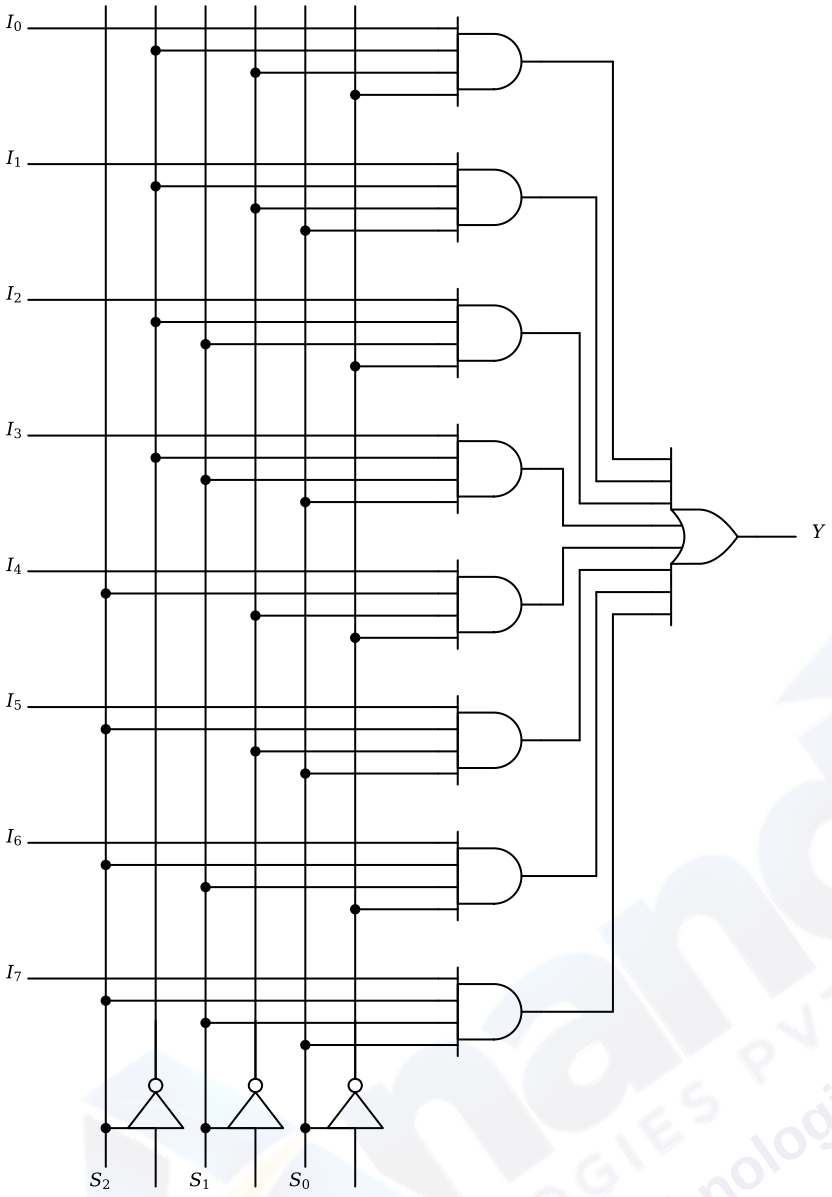
```
Y = AB + AB + AC + BB + BC   (distributive law)
  = AB + AC + B + BC         (AB + AB = AB, BB = B)
  = AB + AC + B(1 + C)
  = AB + AC + B              (1 + C = 1)
  = B(A + 1) + AC
  = B + AC                   (A + 1 = 1)
```

Both results were checked with truth tables (all combinations equal): $Y_i = A + B$, $Y_{ii} = B + AC$.

(Theorems 1 M, each reduction 1½ M)

b) 8 : 1 multiplexer. 4 M

An 8 : 1 MUX has 8 data inputs I_0–I_7, 3 select lines $S_2S_1S_0$ and one output Y. Each AND gate is enabled by one select combination.



S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Logic diagram of 8 : 1 MUX

$$Y = \overline{S}_2\overline{S}_1\overline{S}_0 I_0 + \overline{S}_2\overline{S}_1 S_0 I_1 + \ldots + S_2 S_1 S_0 I_7$$

(Diagram 2 M, truth table 2 M)

c) Pipelining in 8086. 4 M

- **Pipelining** means overlapping the fetch of the next instructions with the execution of the current instruction.
- In 8086 the **BIU** pre-fetches up to 6 instruction bytes into the **instruction queue** whenever the bus is free, while the **EU** decodes and executes the instruction taken from the queue.
- Thus fetch and execute take place at the same time and the EU normally does not have to wait for an instruction to be fetched.
- When a jump / call / loop transfers control, the queue is flushed (emptied) and filling starts again from the new address.

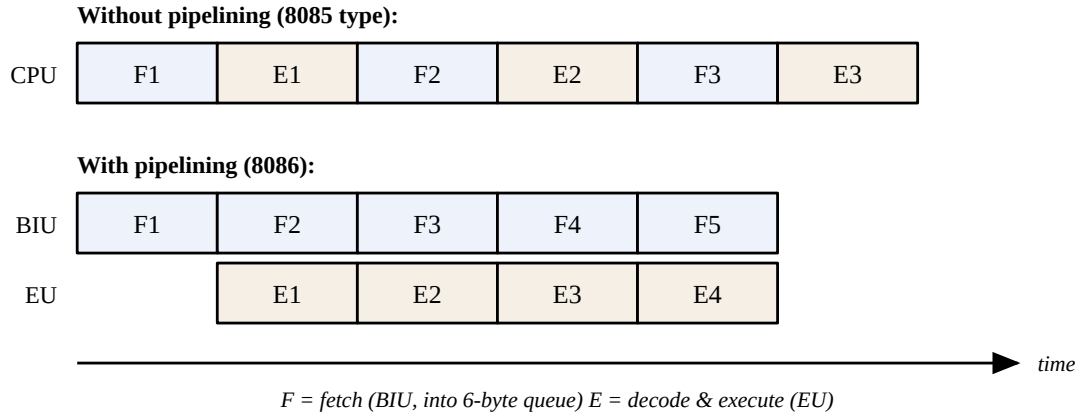


Fig.: Pipelining in 8086 — BIU fetches next instructions while EU executes

Advantages:

1. Faster execution — more instructions per unit time (higher throughput).
2. Better use of the system bus; EU idle time is reduced.
3. BIU and EU work in parallel — fetch overlap hides the memory access time.

(Explanation 2 M, diagram 1 M, advantages 1 M)

d) ALP — concatenation of two strings. 4 M

```
DATA SEGMENT
    STR1 DB 'MICRO'
    LEN1 DB 05H
    STR2 DB 'PROCESSOR'
    LEN2 DB 09H
    RES  DB 20 DUP('$')
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, ES:DATA
START: MOV AX, DATA
       MOV DS, AX
       MOV ES, AX          ; ES = DS for string instructions
       CLD                 ; auto-increment
       LEA SI, STR1        ; copy first string
       LEA DI, RES
       MOV CL, LEN1
       MOV CH, 00H
       REP MOVSB
       LEA SI, STR2        ; append second string
       MOV CL, LEN2        ; DI already points after STR1
       REP MOVSB
       MOV BYTE PTR [DI], '$'
       LEA DX, RES         ; display result
       MOV AH, 09H
       INT 21H
       MOV AH, 4CH
       INT 21H
CODE ENDS
END START
```

Result (checked by running the program on an 8086 simulator):
STR1 = 'MICRO', STR2 = 'PROCESSOR' → RES = '**MICROPROCESSOR**'
Screen output: `MICROPROCESSOR`

(Data 1 M, copying STR1 1 M, appending STR2 2 M)

Q.3

a) Decimal to octal and hexadecimal. 4 M

Steps: divide the decimal number repeatedly by the new base (8 or 16); note the remainders; the remainders read from bottom (last) to top (first) give the answer. In hex, remainders 10–15 are written as A–F.

i) $(583)_{10}$ → octal

Integer part (583): divide repeatedly by 8, write remainders from bottom to top.

Division	Quotient	Remainder
583 ÷ 8	72	7
72 ÷ 8	9	0
9 ÷ 8	1	1
1 ÷ 8	0	1

$\therefore (583)_{10} = (1107)_8$

ii) $(583)_{10}$ → hexadecimal

Integer part (583): divide repeatedly by 16, write remainders from bottom to top.

Division	Quotient	Remainder
583 ÷ 16	36	7
36 ÷ 16	2	4
2 ÷ 16	0	2

$\therefore (583)_{10} = (247)_{16}$

Check: $(1107)_8 = 583$, $(247)_{16} = 583$ ✓

(2 M each)

b) K-map — POS form. 4 M

The given maxterms are the 0s of F; all other cells are 1. For POS, the 0s are grouped.

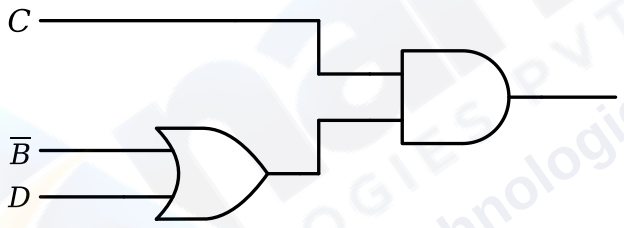
CD / AB	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	0	0	1	0
10	0	0	1	1

K-map for F (grouping 0s)

For POS, 0s are grouped; in each group a variable that is 0 is written as it is and a variable that is 1 is complemented (sum term).

Group	Cells (maxterms)	Sum term
Group 1	0, 1, 4, 5, 8, 9, 12, 13	C
Group 2	4, 6, 12, 14	($\overline{B}$ + D)

Minimal POS: F = C ($\overline{B}$ + D)



Logic diagram (OR–AND)

(K-map 1 M, grouping 1 M, expression 1 M, diagram 1 M)

c) Minimum mode vs maximum mode. 4 M

Minimum mode	Maximum mode
MN/$\overline{MX}$ pin = 1 (connected to +5 V).	MN/$\overline{MX}$ pin = 0 (connected to ground).
Single-processor system (8086 alone).	Multiprocessor system (8086 with 8087 / 8089 coprocessors).
8086 itself generates control signals ALE, M/$\overline{IO}$, $\overline{RD}$, $\overline{WR}$, $\overline{DEN}$, DT/$\overline{R}$, $\overline{INTA}$.	Bus controller 8288 generates the control signals from status lines $\overline{S2}$, $\overline{S1}$, $\overline{S0}$.
HOLD and HLDA are used for DMA / bus request.	$\overline{RQ}$/$\overline{GT0}$ and $\overline{RQ}$/$\overline{GT1}$ are used for bus request / grant.

(Any four points, 1 M each)

d) MOVSW, CMPSW, STOSB, REP. 4 M

1. **MOVSW** (move string word): copies a word from DS:SI to ES:DI, then SI and DI change by 2 (+2 if DF = 0, −2 if DF = 1). No flags affected.
   e.g. `MOV CX, 3 / CLD / REP MOVSW` copies 3 words (1111H, 2222H, 3333H) from S1 to S2 (checked by simulation).
2. **CMPSW** (compare string word): computes [DS:SI] − [ES:DI] (words), sets the flags only, SI and DI change by 2.
   e.g. `REPE CMPSW` compares two word arrays; ZF = 1 at the end means all words are equal.
3. **STOSB** (store string byte): stores AL into ES:DI, then DI ± 1.
   e.g. `MOV AL, '*' / MOV CX, 5 / REP STOSB` fills 5 bytes with '*' (2AH).
4. **REP prefix:** repeats the following string instruction CX times (CX decremented each time, stops when CX = 0). REPE/REPZ also stops when ZF = 0, REPNE/REPNZ stops when ZF = 1 (used with CMPS / SCAS).

(1 M each)

Q.4

a) Output of given logic circuit. 4 M

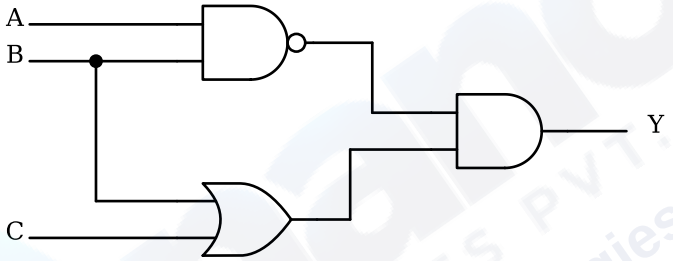


Fig. No. 1 (given)

Gate 1 (NAND): $\overline{AB}$ Gate 2 (OR): B + C Gate 3 (AND): **Y = $\overline{AB}$ · (B + C)** = ($\overline{A}$ + $\overline{B}$)(B + C) = $\overline{A}$B + $\overline{A}$C + $\overline{B}$C

A	B	C	$\overline{AB}$	B + C	Y
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	0	1	0

(Expression 2 M, truth table 2 M)

b) BCD addition; 2's complement subtraction. 4 M

i) $(57)_{10}$ + $(68)_{10}$ in BCD

Write each number in BCD and add digit-wise from the LSD. If a digit sum is greater than 9 (1001) or produces a carry, add 6 (0110) to correct it and pass the carry to the next digit.

$(57)_{10}$ = 0101 0111

$(68)_{10}$ = 0110 1000

Digit	BCD of 1st	BCD of 2nd	Carry in	Binary sum	Correction?	Result digit	Carry out
10^0	0111	1000	0	01111 (15)	Yes, +0110	0101	1
10^1	0101	0110	1	01100 (12)	Yes, +0110	0010	1

$\therefore$ 57 + 68 = 125 = $(0001\ 0010\ 0101)_{BCD}$

ii) $(01011001)_2 - (01110011)_2$ (= 89 − 115)

2's complement of subtrahend 01110011 = 10001100 + 1 = **10001101**. Add it to the minuend:

```
  01011001
+ 10001101
-----------
 11100110
```

No carry → result is **negative**, in 2's complement form. 2's complement of 11100110 = 00011010.
Result = $-(00011010)_2 = -26$

(2 M each)

c) Memory segmentation; physical address. 4 M

- 8086 memory (1 MB) is divided into segments of maximum 64 KB: code, data, stack and extra segments, pointed to by CS, DS, SS and ES.
- A segment register holds the upper 16 bits of the 20-bit starting address of the segment; every location is addressed as segment : offset.
- Segments start at 16-byte boundaries and can overlap. Advantages: 1 MB addressing with 16-bit registers, relocatable programs, separation of code / data / stack.

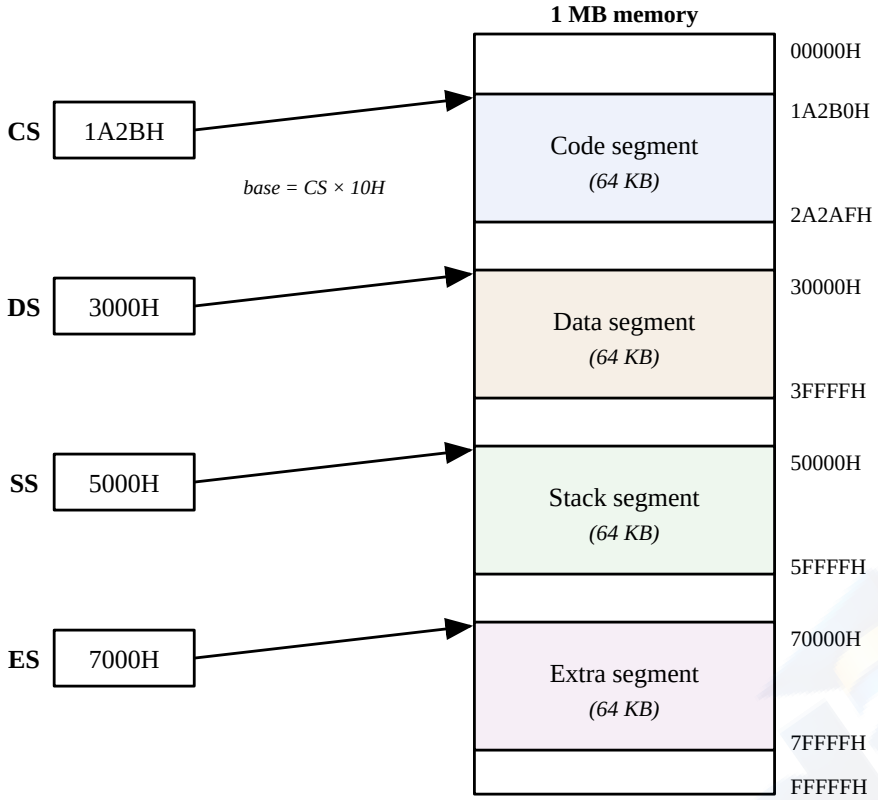


Fig.: Memory segmentation (CS = 1A2BH shown)

Physical address for CS = 1A2BH, IP = 3C4DH:

```
  CS × 10H =  1A2B0H
+ IP       =  03C4DH
------------------------
  PA       =  1DEFDH
```

(Segmentation 2 M, diagram 1 M, calculation 1 M)

d) PUSH, IDIV, XCHG, LOOP. 4 M

1. **PUSH source** (16-bit register / memory / segment register): SP ← SP − 2, then the word is stored at SS:SP (low byte at lower address).
   e.g. SP = 2000H, AX = 1234H → `PUSH AX` → SP = 1FFEH, [1FFEH] = 34H, [1FFFH] = 12H.
2. **IDIV source** (signed division): 8-bit — AX ÷ source → AL = quotient, AH = remainder; 16-bit — DX:AX ÷ source → AX = quotient, DX = remainder.
   e.g. AX = FF9CH (−100), BL = 07H → `IDIV BL` → AL = F2H (−14), AH = FEH (−2).
3. **XCHG destination, source:** exchanges the contents of two registers or a register and memory (not two memory locations, not segment registers).
   e.g. AX = 1111H, BX = 2222H → `XCHG AX, BX` → AX = 2222H, BX = 1111H.
4. **LOOP label:** CX ← CX − 1; if CX ≠ 0 jump to label, else continue. Used for counted loops.
   e.g. `MOV CX, 05H / MOV AL, 00H / BACK: ADD AL, 02H / LOOP BACK` → AL = 0AH (loop executed 5 times).

(1 M each)

e) ALP — count occurrence of a number. 4 M

```
DATA SEGMENT
    ARR   DB 12H, 35H, 12H, 7AH, 0C1H, 12H, 44H, 90H, 12H, 5BH
    KEY   DB 12H          ; number to be counted
    COUNT DB ?
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
       MOV DS, AX
       LEA SI, ARR
       MOV CX, 10         ; 10 elements
       MOV BL, 00H        ; counter = 0
       MOV AL, KEY
UP:    CMP AL, [SI]
       JNE NEXT
       INC BL             ; match found
NEXT:  INC SI
       LOOP UP
       MOV COUNT, BL
       MOV AH, 4CH
       INT 21H
CODE ENDS
END START
```

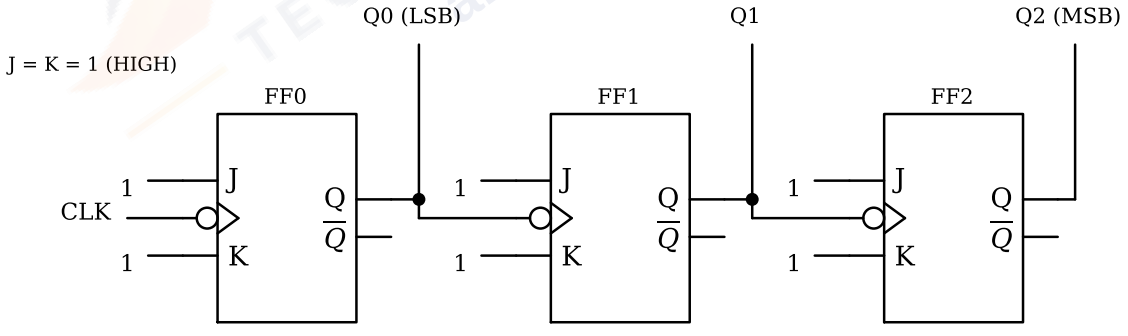
Result (checked by running the program on an 8086 simulator):
Array: 12H, 35H, 12H, 7AH, C1H, 12H, 44H, 90H, 12H, 5BH, KEY = 12H → **COUNT = 04H**

(Initialisation 1 M, compare & count loop 2 M, result 1 M)

Q.5

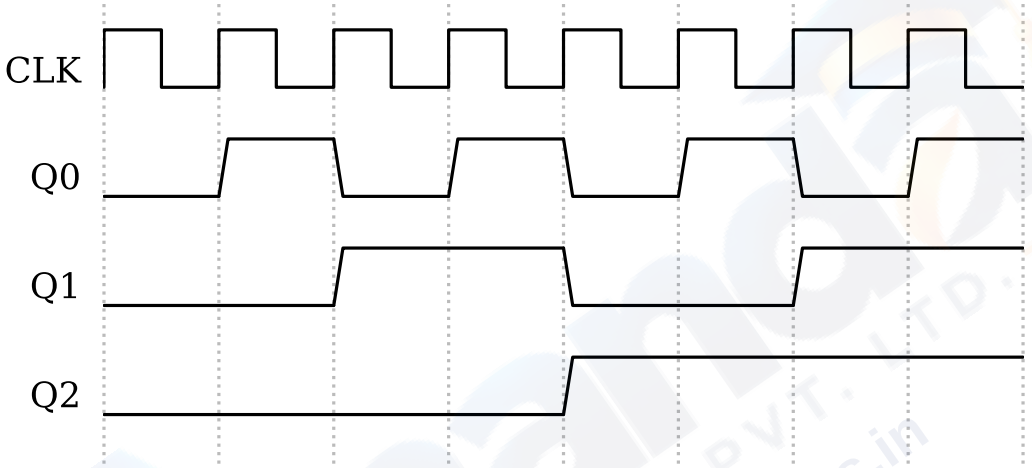
a) 3-bit ripple up counter. 6 M

Three J-K flip-flops with J = K = 1 (toggle mode). The external clock drives FF0; Q of each flip-flop is the clock of the next one (negative-edge triggered). A 3-bit counter has $2^3 = 8$ states (MOD-8): 000 → 111 → 000.



3-bit asynchronous (ripple) up counter

Clock pulse	Q_2	Q_1	Q_0	Count
Initial	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7
8	0	0	0	0



Timing diagram (FF toggles on falling edge of its clock)

Q_0 toggles on every falling clock edge, Q_1 on every falling edge of Q_0 and Q_2 on every falling edge of Q_1; so Q_0, Q_1, Q_2 have frequencies f/2, f/4, f/8.

(Logic diagram 2 M, truth table 2 M, timing diagram 2 M)

b) Minimum mode configuration of 8086. 6 M

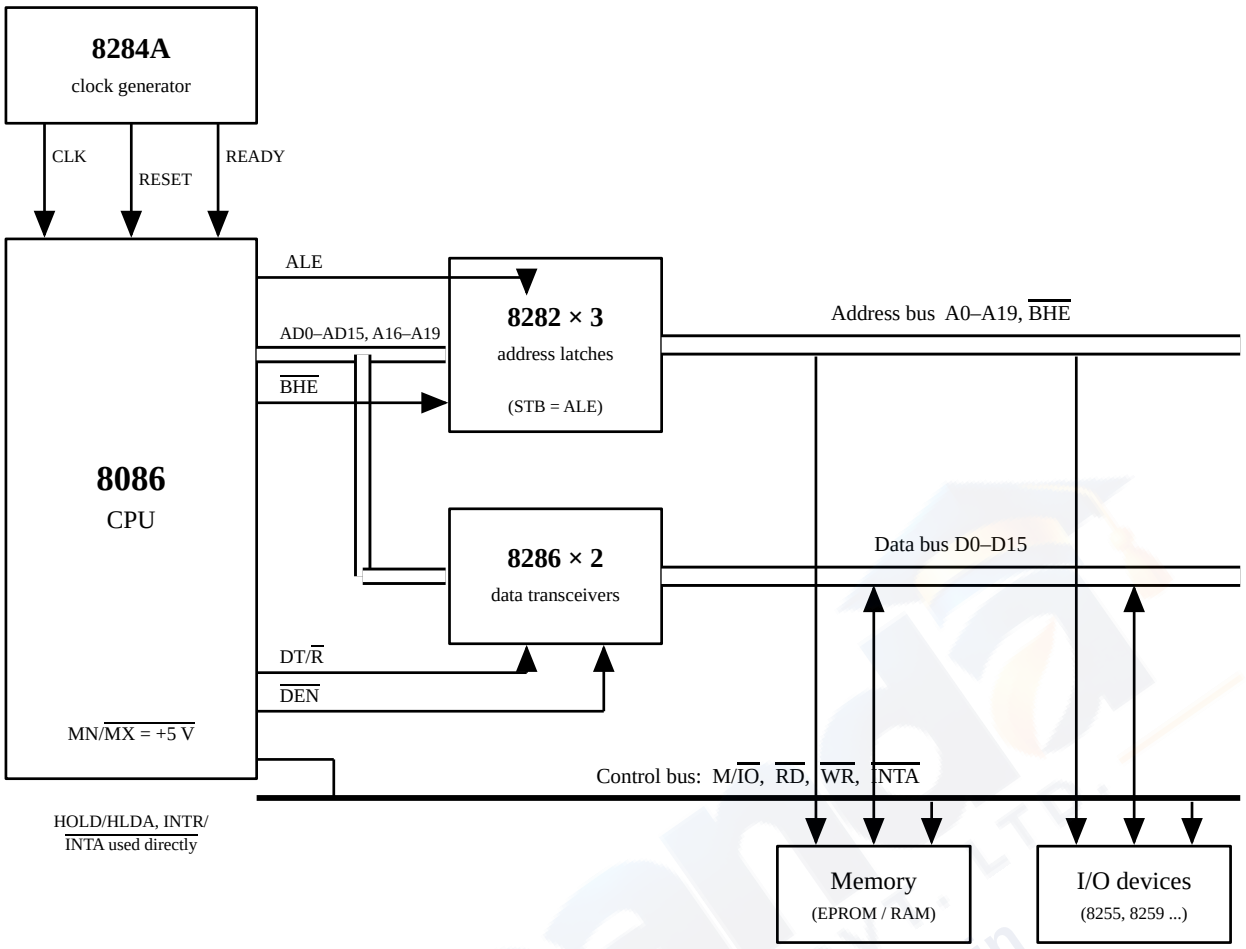
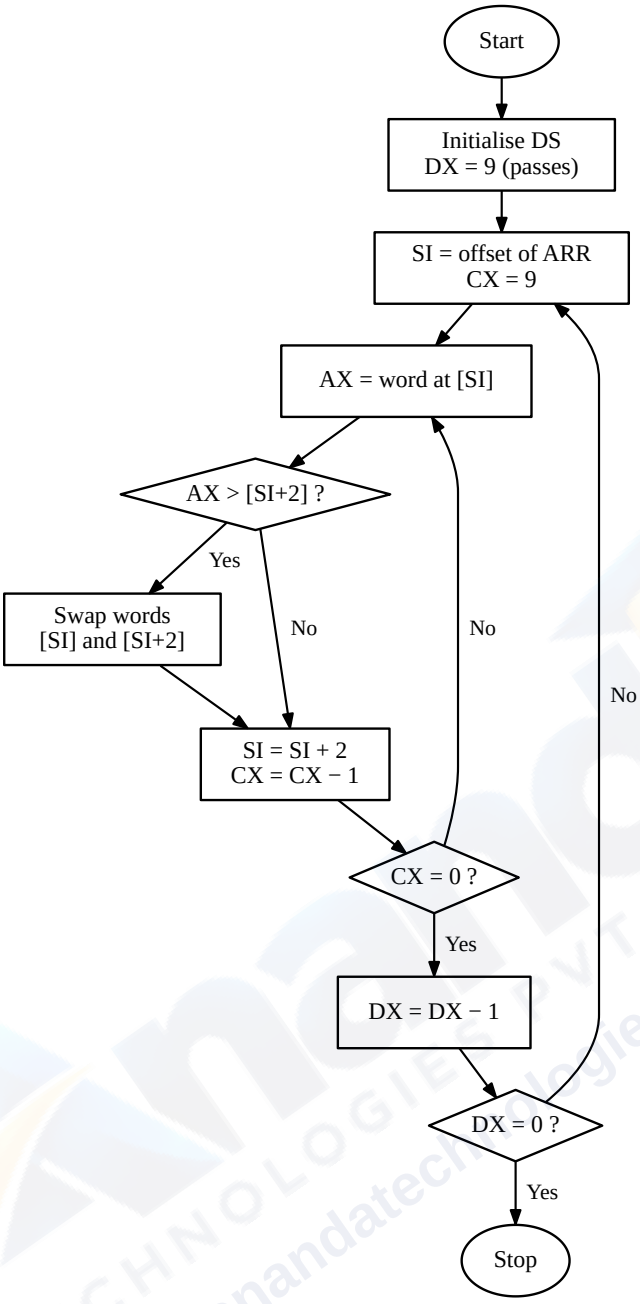


Fig.: Minimum mode configuration of 8086 (MN/MX pin = 1, single-processor system)

1. In minimum mode the MN/$\overline{MX}$ pin is tied to +5 V. The 8086 is the only processor and generates all control signals itself, so no bus controller is needed.
2. **Clock generator 8284A:** produces the CLK (5/8/10 MHz), and synchronises RESET and READY inputs for the CPU.
3. **Address latches 8282 (3 nos.):** AD0–AD15, A16–A19 and $\overline{BHE}$ are multiplexed; ALE (address latch enable) strobes the address into the latches in T1, giving a separate 20-bit address bus.
4. **Data transceivers 8286 (2 nos.):** buffer the 16-bit data bus. DT/$\overline{R}$ selects direction (1 = transmit / write, 0 = receive / read) and $\overline{DEN}$ enables the transceivers during data transfer.
5. **Control signals:** M/$\overline{IO}$ (memory or I/O), $\overline{RD}$, $\overline{WR}$ (read / write strobes) and $\overline{INTA}$ (interrupt acknowledge) are given directly to memory and I/O.
6. **HOLD / HLDA** are used by a DMA controller to request and get the bus; INTR / NMI are interrupt inputs.

(Diagram 3 M, explanation 3 M)

c) ALP — ascending order of 16-bit numbers. 6 M



Flowchart — ascending order of 16-bit numbers

```
DATA SEGMENT
    ARR DW 4A2FH, 1C03H, 9E77H, 0B12H, 7D40H, 2F9AH, 0E5C1H, 5066H, 3B8DH, 8124H
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV DX, 9           ; passes
OUTER:  LEA SI, ARR
        MOV CX, 9           ; comparisons per pass
INNER:  MOV AX, [SI]
        CMP AX, [SI+2]      ; compare adjacent words
        JBE SKIP            ; in order -> no swap
        XCHG AX, [SI+2]     ; swap words
        MOV [SI], AX
```

```
SKIP:   ADD SI, 2            ; next word (2 bytes)
        LOOP INNER
        DEC DX
        JNZ OUTER
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

Result (checked by running the program on an 8086 simulator):
Before: 4A2FH, 1C03H, 9E77H, 0B12H, 7D40H, 2F9AH, E5C1H, 5066H, 3B8DH, 8124H
After: 0B12H, 1C03H, 2F9AH, 3B8DH, 4A2FH, 5066H, 7D40H, 8124H, 9E77H, E5C1H

Words occupy 2 bytes, so SI is increased by 2 and the next number is at [SI + 2].

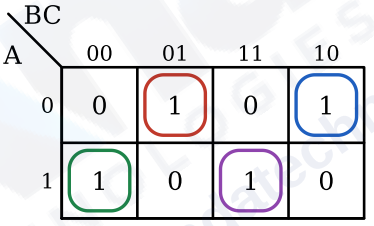
(Flowchart 2 M, program 4 M)

Q.6

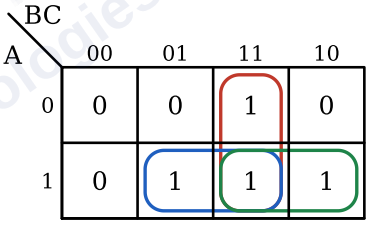
a) Full adder design. 6 M

A full adder adds three bits A, B and C_{in} (carry from previous stage) and gives Sum S and Carry C_{out}. (In the K-maps C = C_{in}.)

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



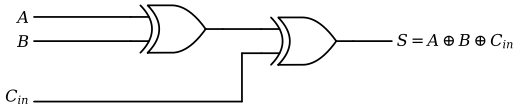
S = A'B'C + A'BC' + AB'C' + ABC



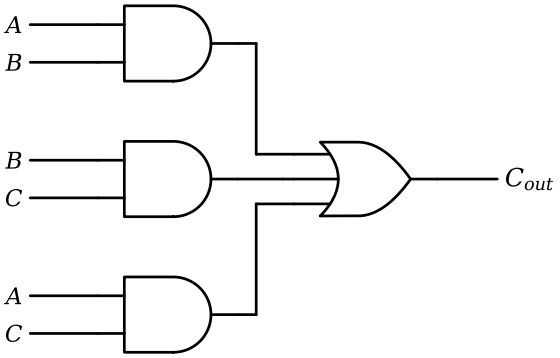
Cout = BC + AC + AB

The 1s of S do not form any group (checker-board pattern), so S = $\overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$ = **A ⊕ B ⊕ C_{in}**

$C_{out} = AB + BC_{in} + AC_{in}$ (three pairs in the K-map).

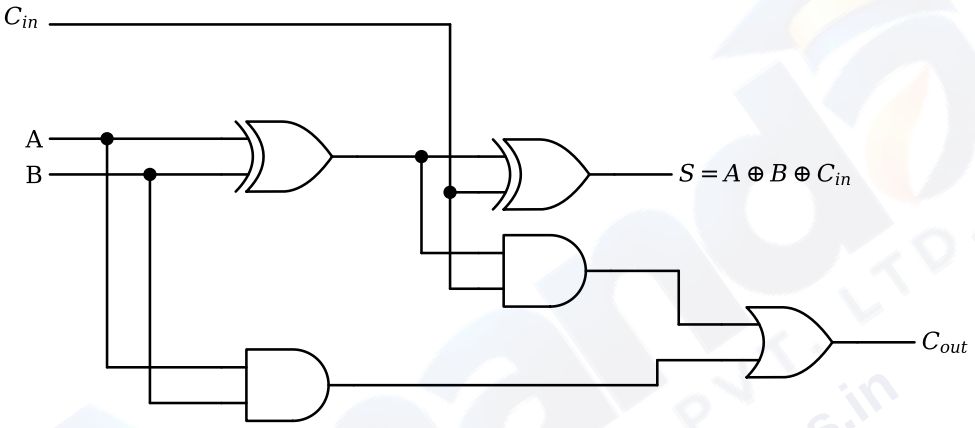


Sum: two EX-OR gates



Carry: AND–OR (C = Cin)

Full adder using two half adders: HA-1 adds A and B ($S_1 = A \oplus B$, $C_1 = AB$); HA-2 adds S_1 and C_{in} ($S = S_1 \oplus C_{in}$, $C_2 = S_1C_{in}$); $C_{out} = C_1 + C_2 = AB + C_{in}(A \oplus B)$.



Full adder using two half adders and an OR gate

(Truth table 1 M, K-maps 2 M, logic diagram 1 M, using half adders 2 M)

b) AL and flags after DAA sequence. 6 M

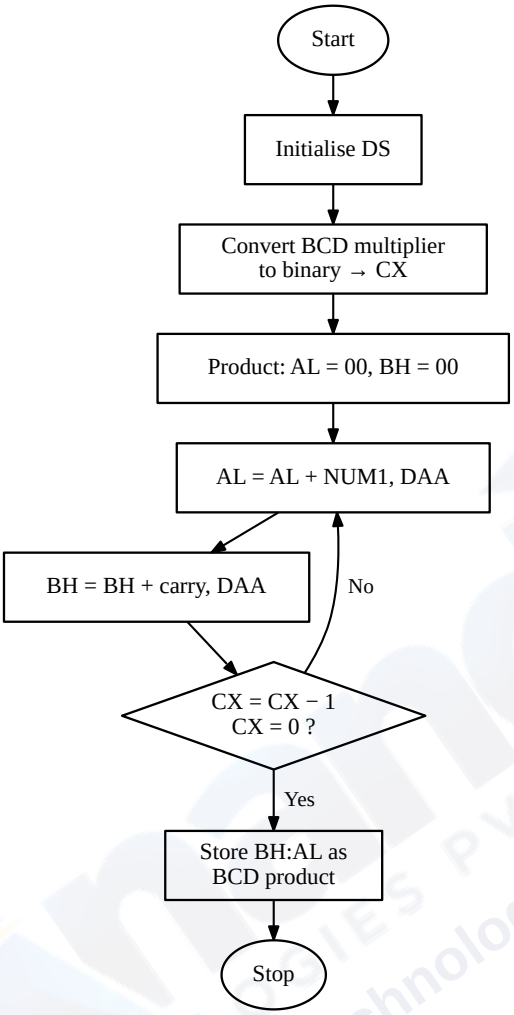
DAA rule: if lower nibble of AL > 9 or AF = 1 → add 06H (AF = 1); if AL (before) > 99H or CF = 1 → add 60H (CF = 1). MOV does not change flags (CF, AF, ZF are 0 at start).

Instruction	Result	CF	AF	ZF	Remark
`MOV AL, 79H`	AL = 79H	0	0	0	load
`ADD AL, 25H`	AL = 9EH	0	0	0	79H + 25H = 9EH (binary); lower nibble E > 9
`DAA`	AL = 04H	1	1	0	E > 9 → +06H (AF = 1); 9EH > 99H → +60H (CF = 1). Decimal: 79 + 25 = 104 → AL = 04, CF = 1
`MOV BL, 47H`	AL = 04H	1	1	0	load BL (AL unchanged)
`ADD AL, BL`	AL = 4BH	0	0	0	04H + 47H = 4BH, no carries
`DAA`	AL = 51H	0	1	0	B > 9 → +06H → 51H; no decimal carry (04 + 47 = 51)

(1 M each instruction)

c) ALP — multiplication of two BCD numbers. 6 M

Method: the BCD multiplicand is added to the product as many times as the value of the multiplier; DAA after every addition keeps the product in BCD. (Also checked with 99 × 99 = 9801 → RESHI = 98H, RESLO = 01H.)



Flowchart — BCD multiplication by repeated addition

```
DATA SEGMENT
    NUM1  DB 25H          ; multiplicand (BCD 25)
    NUM2  DB 12H          ; multiplier   (BCD 12, non-zero)
    RESLO DB ?            ; lower 2 BCD digits of product
    RESHI DB ?            ; higher 2 BCD digits
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
START: MOV AX, DATA
       MOV DS, AX
       MOV AL, NUM2       ; convert BCD multiplier to binary count
       MOV BL, AL
       AND BL, 0FH        ; BL = units digit
       MOV CL, 04H
       SHR AL, CL         ; AL = tens digit
       MOV DL, 0AH
       MUL DL             ; AX = tens x 10
       ADD AL, BL         ; AL = multiplier in binary
       MOV CL, AL
       MOV CH, 00H        ; CX = loop count
```

```
        MOV AL, 00H         ; product low  (BCD)
        MOV BH, 00H         ; product high (BCD)
BACK:   ADD AL, NUM1        ; add multiplicand
        DAA                 ; adjust low byte
        XCHG AL, BH         ; (XCHG does not change CF)
        ADC AL, 00H         ; add BCD carry to high byte
        DAA
        XCHG AL, BH
        LOOP BACK           ; repeated addition
        MOV RESLO, AL
        MOV RESHI, BH
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

Result (checked by running the program on an 8086 simulator):
NUM1 = 25 (BCD), NUM2 = 12 (BCD) → **RESHI = 03H, RESLO = 00H** → product = 0300 (25 × 12 = 300)

(Flowchart 2 M, program 4 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in