

Μικροϋπολογιστές

1η Ομάδα Ασκήσεων

Παπαρρηγόπουλος Θωδωρής el18040 paparrigopoulosthodoris@gmail.com

Φιλιππόπουλος Ορφέας el18082 orfeasfil2000@gmail.com

1η Άσκηση

Βασισμένοι στον πίνακα 2 στην σελίδα 98-99, η αναπαράσταση της γλώσσας μηχανής
0E 08 3A 00 20 17 DA 0D 08 0D C2 05 08 79 2F 32 00 30 CF, αντιστοιχίζεται στο παρακάτω
πρόγραμμα assembly.

0800	0E	MVI C, 08H	0800	0E	0801	08	0802	3A	0803	00	0804	20	0805	17	0806	DA	0807	0D	0808
0801	08		0809	C2	080B	05	080C	08	080D	79	080E	2F	080F	32	0810	00	0811	30	0812
0802	3A	LDA 2000H	0814	00	0815	00	0816	00	0817	00	0818	00	0819	00	081A	00	081B	00	081C
0803	00		081E	00	081F	00	0820	00	0821	00	0822	00	0823	00	0824	00	0825	00	0826
0804	20		0828	00	0829	00	082A	00	082B	00	082C	00	082D	00	082E	00	082F	00	0830
			0832	00	0833	00	0834	00	0835	00	0836	00	0837	00	0838	00	0839	00	083A
			083C	00	083D	00	083E	00	083F	00	0840	00	0841	00	0842	00	0843	00	0844
			0846	00	0847	00	0848	00	0849	00	084A	00	084B	00	084C	00	084D	00	084E
			0850	00	0851	00	0852	00	0853	00	0854	00	0855	00	0856	00	0857	00	0858
			085A	00	085B	00	085C	00	085D	00	085E	00	085F	00	0860	00	0861	00	0862
			0864	00	0865	00	0866	00	0867	00	0868	00	0869	00	086A	00	086B	00	086C
			086E	00	086F	00	0870	00	0871	00	0872	00	0873	00	0874	00	0875	00	0876
			0878	00	0879	00	087A	00	087B	00	087C	00	087D	00	087E	00	087F	00	0880
			0882	00	0883	00	0884	00	0885	00	0886	00	0887	00	0888	00	0889	00	088A
			088C	00	088D	00	088E	00	088F	00	0890	00	0891	00	0892	00	0893	00	0894
			0896	00	0897	00	0898	00	0899	00	089A	00	089B	00	089C	00	089D	00	089E
			08A0	00	08A1	00	08A2	00	08A3	00	08A4	00	08A5	00	08A6	00	08A7	00	08A8
			08AA	00	08AB	00	08AC	00	08AD	00	08AE	00	08AF	00	08B0	00	08B1	00	08B2
			08B4	00	08B5	00	08B6	00	08B7	00	08B8	00	08B9	00	08BA	00	08BB	00	08BC
			08BE	00	08BF	00	08C0	00	08C1	00	08C2	00	08C3	00	08C4	00	08C5	00	08C6
			08C8	00	08C9	00	08CA	00	08CB	00	08CC	00	08CD	00	08CE	00	08CF	00	08D0
			08D2	00	08D3	00	08D4	00	08D5	00	08D6	00	08D7	00	08D8	00	08D9	00	08DA
			08DC	00	08DD	00	08DE	00	08DF	00	08E0	00	08E1	00	08E2	00	08E3	00	08E4
			08E6	00	08E7	00	08E8	00	08E9	00	08EA	00	08EB	00	08EC	00	08ED	00	08EE
			08F0	00	08F1	00	08F2	00	08F3	00	08F4	00	08F5	00	08F6	00	08F7	00	08F8
			08FA	00	08FB	00	08FC	00	08FD	00	08FE	00	08FF	00	0900	00	0901	00	0902
			0904	00	0905	00	0906	00	0907	00	0908	00	0909	00	090A	00	090B	00	090C
			090E	00	090F	00	0910	00	0911	00	0912	00	0913	00	0914	00	0915	00	0916
			0918	00	0919	00	091A	00	091B	00	091C	00	091D	00	091E	00	091F	00	0920
			0922	00	0923	00	0924	00	0925	00	0926	00	0927	00	0928	00	0929	00	092A
			092C	00	092D	00	092E	00	092F	00	0930	00	0931	00	0932	00	0933	00	0934
			0936	00	0937	00	0938	00	0939	00	093A	00	093B	00	093C	00	093D	00	093E
			0940	00	0941	00	0942	00	0943	00	0944	00	0945	00	0946	00	0947	00	0948
			094A	00	094B	00	094C	00	094D	00	094E	00	094F	00	0950	00	0951	00	0952
			0954	00	0955	00	0956	00	0957	00	0958	00	0959	00	095A	00	095B	00	095C
			095E	00	095F	00	0960	00	0961	00	0962	00	0963	00	0964	00	0965	00	0966
			0968	00	0969	00	096A	00	096B	00	096C	00	096D	00	096E	00	096F	00	0970

RESET RUN

HOWR STEP INSTR STEP

INTRPT FETCH PC

FETCH ADDR FETCH REG

DECR STORE/ INCR

D E F

A B C

7 8 9

4 5 6

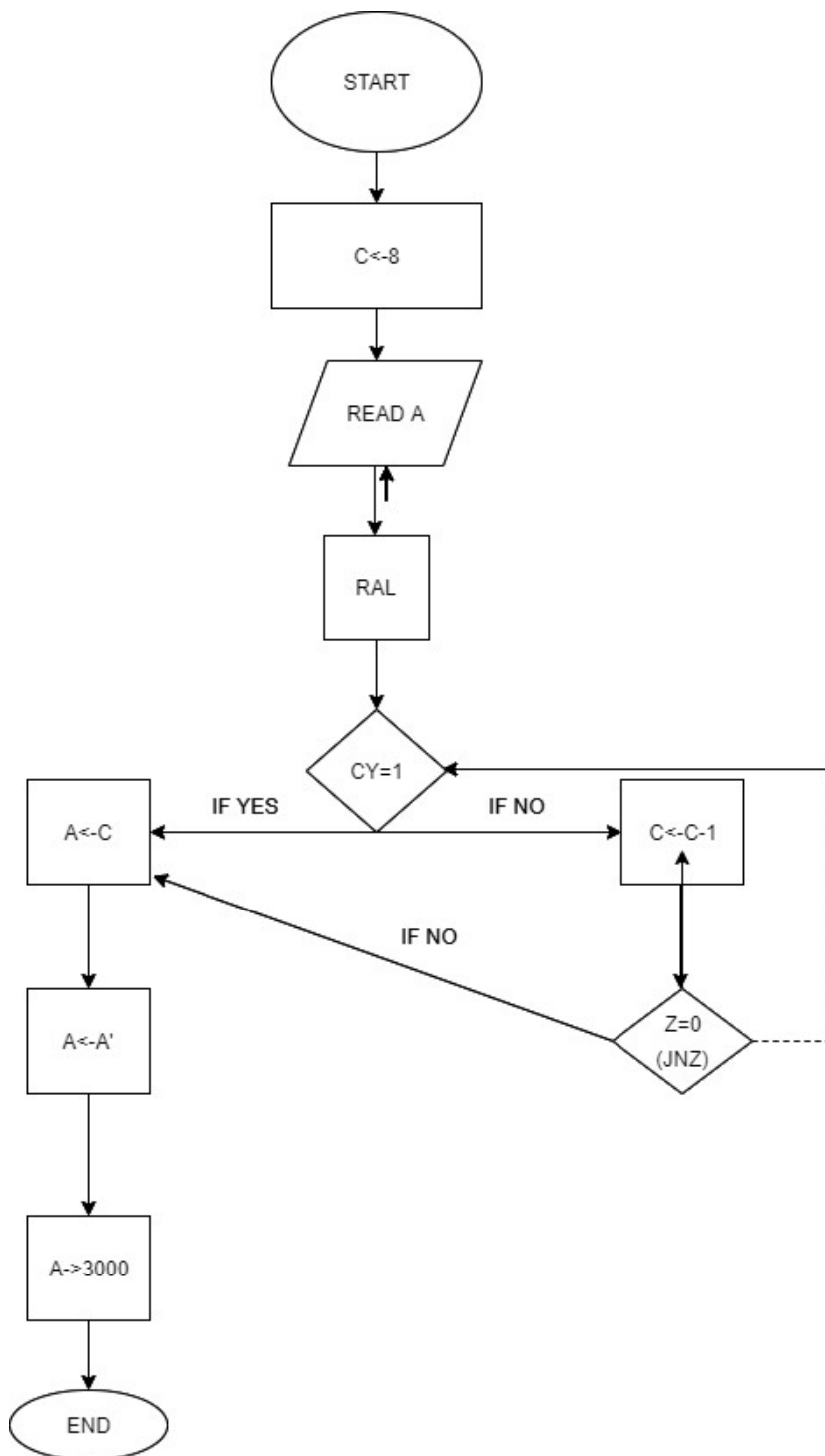
0 1 2 3

0E 08 3A 00 20 17 DA 0D 08 0D C2 05 08 79 2F 32 00 30 CF

Με τη βοήθεια του προσομοιωτή συμπεράναμε ότι η λειτουργία του προγράμματος είναι να βρίσκει τη θέση του αριστερότερου 1 σε έναν 8-bit αριθμό που δέχεται ως είσοδο (αν η είσοδος δεν έχει καθόλου 1, τότε τυπώνει 0).

Για να γίνεται επαναληπτικά αυτή η διαδικασία αρκεί να βάλουμε μια ετικέτα στην αρχή START και στο τέλος να προσθέσουμε μια εντολή JMP START (πριν το END). Σε αυτή την περίπτωση το RST 1, χρησιμοποιείται σαν μία παύση του προγράμματος ώστε να εισάγουμε έναν νέο αριθμό στη θέση μνήμης που θα φορτώσουμε στην επόμενη επανάληψη. Πατώντας RUN το πρόγραμμα ξεκινάει από την αρχή.

Το πρόγραμμα ροής είναι:



2η Άσκηση

Ουσιαστικά, για αυτήν την άσκηση έχουμε 3 πιθανές καταστάσεις. Ή θα κινηθεί το λαμπάκι προς τα αριστερά ή προς τα δεξιά ή να μείνει στάσιμο. Σώζουμε την εκάστοτε τιμή στον register D και τον επικαλούμαστε κάθε φορά που είναι ανάψουμε το αντίστοιχο λαμπάκι.
Για την επεξηγηματική ανάλυση της άσκησης έχουμε βάλει τα αντίστοιχα σχόλια.

```
IN 10H
LXI B,01F4H      ; 500
MVI A,01H
```

```
PRINT:
CMA              ; Inverse logic at 3000H
STA 3000H        ; store A in 3000H
CMA              ; restore value
MOV D,A          ; save to A to D
```

```
START:
CALL DELB        ; pause for 1ms * B = 1ms * 500 = 0.5 s
LDA 2000H        ; load 2000H to A
ANI 81H          ; A = A AND 10000001 (To keep MSB and LSB)
CPI 01H          ; if A equals 00000001 then Zero flag = 1 else 0
JZ LEFT          ; if zero flag = 1 jump to LEFT
CPI 81H          ; check now if A equals to 10000001 then zero flag = 1 else 0
JZ RIGHT         ; if zero flag = 1 then jump to RIGHT
JMP START        ; else jump to back to START (we are in a waiting position)
```

```
LEFT:
MOV A,D          ; Save the previous value of the counter (stored in D)
RLC              ; Rotate Left without Carry
JMP PRINT        ; Jump back to PRINT
```

```
RIGHT:
MOV A,D          ; Save the previous value of the counter (stored in D)
RRC              ; Rotate Right without Carry
JMP PRINT        ; Jump back to PRINT
END
```

3η Άσκηση

Στο παρακάτω πρόγραμμα ελέγχουμε εάν ο αριθμός που φορτώσαμε από τη θέση 2000H είναι μεγαλύτερος του 199. Εάν είναι τότε αναβοσβήνουν τα MSB (βάζουμε (στη διεύθυνση 3000H) το 11110000 κάνουμε CALL DELB και στη συνέχεια βάζουμε το 00000000 και ξανακαλούμε CALL DELB) αλλιώς ελέγχουμε εάν είναι μεγαλύτερος του 100. Εάν είναι τότε αναβοσβήνουν τα LSB (παρόμοια διαδικασία με πριν) αλλιώς βρίσκω τον BCD αριθμού και τον βάζω στη θέση 3000H και κάνω CALL DELB προκειμένου να ανάψουν τα Led (δηλαδή να προλάβει να τα δει ανθρώπινο μάτι).

```
START:
LXI B,01F4H      ; B = 500 (500ms delay)
LDA 2000H        ; Load 2000H to A
CPI C8H          ; if A greater than or equals to 200 then Carry flag sets to 0 else 1
JNC CASE2        ; if Carry flag is 0 jump to CASE2
CPI 64H          ; if A greater than or equals to 100 then Carry flag sets to 0 else 1
JNC CASE1        ; if Carry flag is 0 jump to CASE1
MVI B,FFH        ; else set B = 255
```

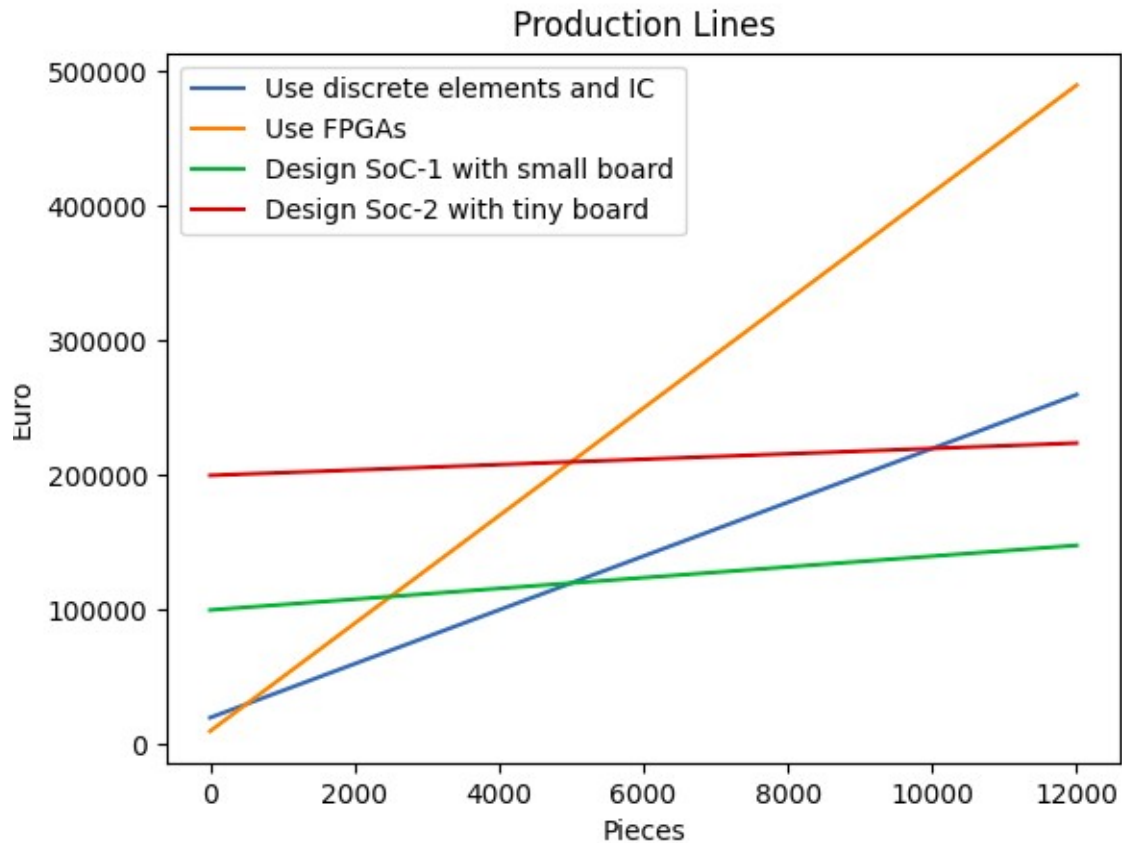
```
DECA:
INR B            ; B = B + 1
SUI 0AH          ; A = A - 10
JNC DECA         ; if carry flag is 1 then jump to DECA
ADI 0AH          ; A = A + 10
MOV C,A          ; C = A
MOV A,B          ; A = B
RLC              ; Rotate accumulator Left without Carry 4 times
RLC              ; This way we keep the 4 MSBs of the bcd number
RLC
RLC
ADD C            ; A = A + C (Add 4 LSBs of the bcd number)
STA 3000H        ; Store at 3000H address the content of A
RST 1            ; system pause
JMP START        ; Jump back to START
```

```
CASE2:
MVI A,F0H        ; A = 240 (11110000)
STA 3000H        ; Store A to 3000H address
CALL DELB        ; wait
MVI A,00H        ; Set A = 0 (00000000)
STA 3000H        ; Store at 3000H address 0
JMP START        ; Jump back to START
```

```
CASE1:
MVI A,0FH        ; A = 15 (00001111)
STA 3000H        ; Store at 3000H 15 (A)
CALL DELB        ; wait
MVI A,00H        ; A = 0 (00000000)
STA 3000H        ; Store at 3000H address 0
JMP START        ; Jump back to START
```

END

4η Άσκηση



Υπολογίζοντας τα σημεία τομής των τριών καμπυλών, βρίσκουμε ποια τεχνολογία συμφέρει για κάθε διάστημα:

- 0 – 500 τεμάχια: 2
- 500 – 5000 τεμάχια: 1
- 5000 – 50000 τεμάχια: 3
- 50000 ή περισσότερα: 4

Για να εξαφανιστεί η επιλογή της 1ης τεχνολογίας θέλουμε στα 5000 τεμάχια (σημείο τομής 1ης και 3ης καμπύλης) η 2 η να είναι ίση με την 1η . Έτσι η καμπύλη 2 θα είναι πάντα κάτω από την 1η για το διάστημα που η 1η ήταν η φθηνότερη επιλογή, επομένως δεν θα επιλεγεί ποτέ.

Άρα στα 5000 τεμάχια πρέπει $10.000 + \text{κόστος} \cdot 5000 = 100.000 + 4 \cdot 5000$, άρα κόστος/τεμάχιο = 22.

Επομένως πρέπει το κόστος των IC = 12 €/τεμάχιο (22 – κόστος πλακέτας και συναρμολόγησης= 10€), για να είναι η πιο συμφέρουσα επιλογή.

Ασκηση 5

i) Gate Level:

primitive for_table(x, A, B, C, D);

output x;

input A, B, C, D;

table

0 0 0 0: 1;

0 0 0 1: 0;

0 0 1 0: 1;

0 0 1 1: 1;

0 1 0 0: 0;

0 1 0 1: 1;

0 1 1 0: 0;

0 1 1 1: 1;

1 0 0 0: 0;

1 0 0 1: 1;

1 0 1 0: 1;

1 0 1 1: 1;

1 1 0 0: 0;

1 1 0 1: 1;

1 1 1 0: 1;

1 1 1 1: 0;

endtable

endprimitive

module Askisi5 (A, B, C, D, E, F1, F2, F3, F4);

Input A, B, C, D, E;

Output F1, F2, F3, F4;

wire Bnot, Cnot, w1, w2, w3, w4;

not

(Bnot, B),

(Cnot, C);

and (w1, B, C);

or (w2, w1, D);

and (w3, w2, A);

and (w4, Bnot, Cnot, D);

or (F1, w3, w4);

for_table (F2, A, B, C, D);

wire w5, w6, w7, w8, w9, w10;

and (w5, A, B, C);

and (w6, B, C);

or (w7, w6, A);

and (w8, w7, D);

or (w9, B, C);

and (w10, w9, D, E);

or (F3, w5, w8, w10);

wire w11, w12, w13, w14;

```

    and (w11, C, D);
    or (w12, w11, B, E);
    and (w13, w12, A);
    and (w14, B, C, D, E);
    or (F4, w13, w14);
endmodule

```

(ii) Dat Flow Level:

```

module Askisi5b (A, B, C, D, E, F1, F2, F3, F4);
    output F1, F2, F3, F4;
    input A, B, C, D, E;
    assign
        F1 = (A & ((B & C) | D)) | (~B & ~C & D);
        F2 = (~A & ~B & ~C & ~D) | (~A & ~B & C & ~D) | (~A & ~B & C & D) | (~A &
B & ~C & D) |
        (~A & B & C & D) | (A & ~B & ~C & D) | (A & ~B & C & ~D) | (A & ~B &
C
        & D) |
        (A & B & ~C & D) | (A & B & C & ~D);
        F3 = (A & B & C) | ((A | (B & C)) & D) | ((B | C) & D & E);
        F4 = (A & (B | (C & D) | E)) | (B & C & D & E);
endmodule

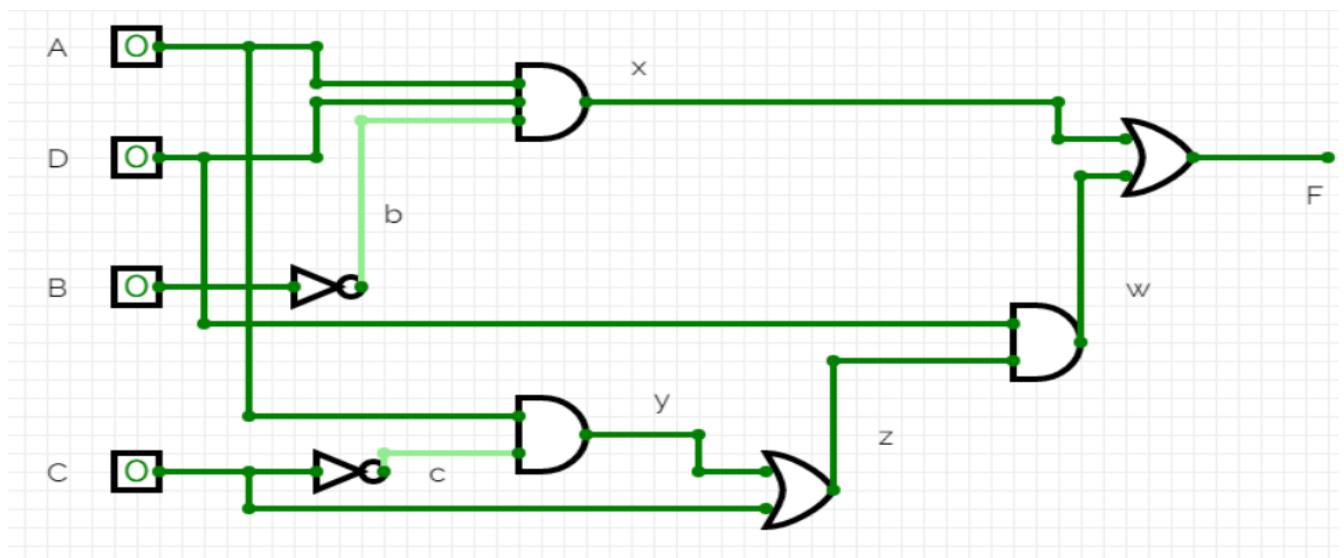
```

Άσκηση 6:

i)

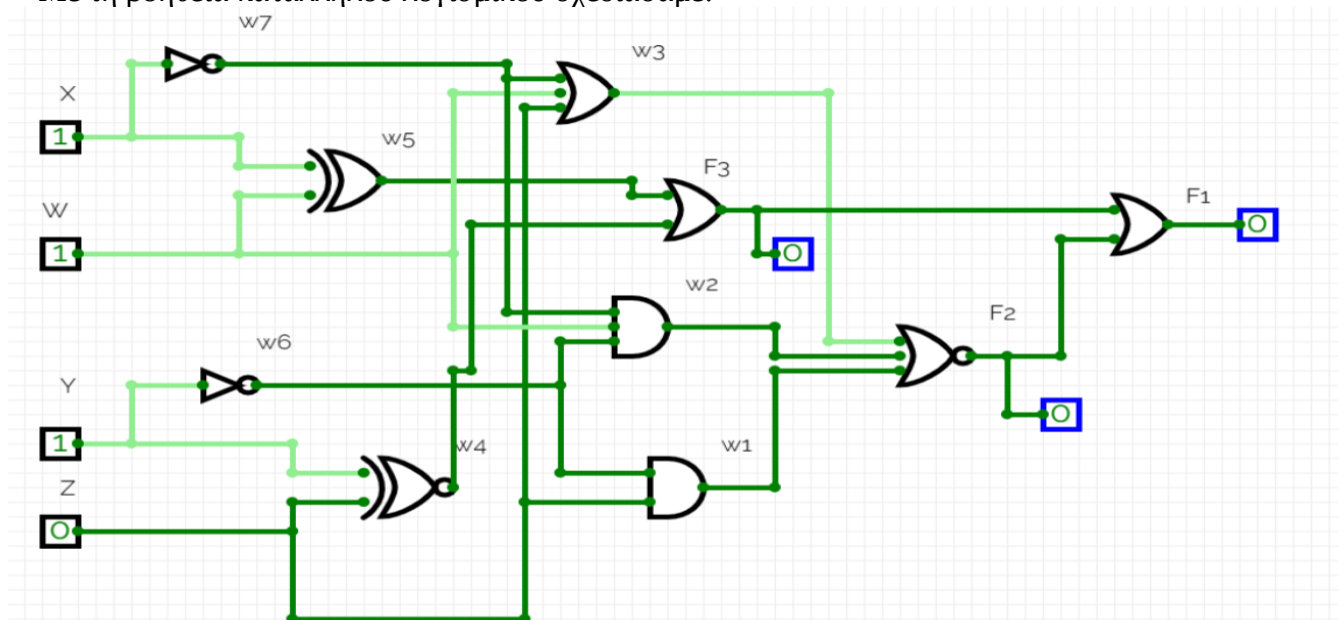
a)

Με τη βοήθεια κατάλληλου λογισμικού σχεδιάσαμε:



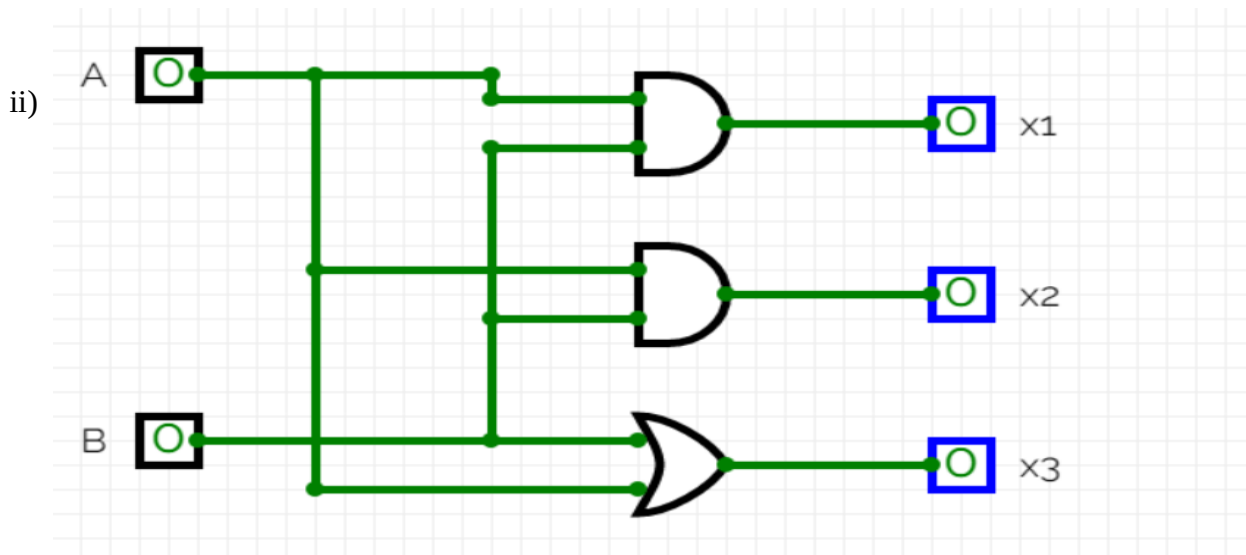
b)

Με τη βοήθεια κατάλληλου λογισμικού σχεδιάσαμε:



c)

Με τη βοήθεια κατάλληλου λογισμικού σχεδιάσαμε:



```
module half_adder(output S, C, input x, y);
    xor(S, x, y);
    and(C, x, y);
endmodule
```

```
module full_adder(output S, C, input x, y, z);
    wire S1, C1, C2;
    half_adder
        HA1(S1, C1, x, y),
        HA2(S, C2, S1, z);
    or G1(C, C2, C1);
endmodule
```

```
module 4_bit_adder_subtractor(output [3: 0] Sum, output C4, input [3:
0] A, B, input C0);
    wire C1, C2, C3;
    wire [3: 0] comp;
```

xor #(εάν C0 = 1 => αφαίρεση => xor το B για συμπλήρωμα ως προς 1 και του προσθέτουμε #και το C0 (που είναι 1) προκειμένου να παράξουμε το συμπλήρωμα ως προς 2 (αφαίρεση), #εάν C0 = 0 τότε 0000 xor B = B και γίνεται κανονικά η πρόσθεση)

```
        (comp[0], B[0], C0),
        (comp[1], B[1], C0),
        (comp[2], B[2], C0),
        (comp[3], B[3], C0);
    full_adder
        FA0(Sum[0], C1, A[0], comp[0], C0),
        FA1(Sum[1], C2, A[1], comp[1], C1),
        FA2(Sum[2], C3, A[2], comp[2], C2),
        FA3(Sum[3], C4, A[3], comp[3], C3);
endmodule
```

iii)

```
module 4_bit_adder_subtractor(output [3: 0] Sum, output C4, input [3: 0] A, B, input C0);
```

```
assign {C4, Sum} = C0 ? (A+ (~B)+1) : (A+B); #με βάση το C0 θα γίνει και η αντίστοιχη
#πράξη, στην αφαίρεση το B σε
#ως προς 2)
συμπλήρωμα
endmodule
```

Άσκηση 7

Τα Mealy διάγραμμα εξαρτώνται και από το current state αλλά και από την είσοδο x και για το λόγο αυτό κάνουμε και το αντίστοιχο assign. Από την άλλη το Moore διάγραμμα εξαρτάται μόνο από το current state και όχι από την είσοδο άμεσα. Η είσοδος απλά μεταφέρει το current state από το ένα state στο άλλο (μπορεί να κάνει και self loop).

```
module Mealy(y, x, clock, reset);  
  
output y;  
input x, clock, reset;  
reg [1: 0] state;  
parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;  
always @ (posedge clock, negedge reset)  
if (reset == 0) state <= a;  
else case (state)  
a: if (~x) state <= d; else state <= a;  
b: if (~x) state <= c; else state <= a;  
c: if (~x) state <= d; else state <= b;  
d: if (~x) state <= c; else state <= d;  
endcase  
assign y = (~state[0] & ~state[1] & ~x) | (state[0] & state[1] & x) | (~state[0] & state[1] & ~x) |  
(state[0] & ~state[1] & ~x);  
endmodule
```

//mealy depends on current state and input

```
i)  
module Mealy(y, x, clock, reset);  
output y;  
input x, clock, reset;  
reg [1: 0] state;  
parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;  
always @ (posedge clock, negedge reset)  
if(reset == 0) state <= a  
else case (state)  
a: if(~x) state<= d else state <= a;  
b: if(~x) state<= c else state <= a;  
c: if(~x) state<= b else state <= d;  
d: if(~x) state<= c; else state <= d;  
Endcase  
assign y = (state[0] & ~state[1]) | (~state[0] & state[1]);  
Endmodule
```

//moore depends on current state only