Σχολή Ηλεκτρολόγων Μηχανικών
και
Μηχανικών Υπολογιστών

Συστήματα Μικροϋπολογιστών [Ροή Υ]

1η Ομάδα Ασκήσεων

Δημήτρης Δήμος

031 17 165

6ο Εξάμηνο

Απρίλιος 2020

1η Άσκηση

Σύμφωνα με τον πίνακα 2 του Παραρτήματος 2 των σημειώσεων "Εισαγωγή στο Εκπαιδευτικό Σύστημα μLab" πραγματοποιήθηκε η διαδικασία της αποκωδικοποίησης του δοθέντος προγράμματος. Οι εντολές γλώσσας μηχανής μεταφράζονται σε **assembly** ως εξής:

Machine Language	⇔	**Assembly 8085**
06 01	⇔	MVI B, 01H
3A 00 20	⇔	LDA 2000H
FE 00	⇔	CPI 00H
CA 13 08	⇔	JZ FIRST
1F	⇔	RAR
DA 12 08	⇔	JC SECOND
04	⇔	INR B
C2 0A 08	⇔	JNZ THIRD
78	⇔	MOV A, B
2F	⇔	CMA
32 00 30	⇔	STA 3000H
CF	⇔	RST 1

Στις εντολές (γλώσσας μηχανής) **CA 13 08**, **DA 12 08** και **C2 0A 08** έχουμε χρησιμοποιήσει ονόματα ετικετών προκειμένου να δώσουμε μια ρεαλιστική εικόνα αναφορικά με τον κώδικα μέσω του οποίου προέκυψε το assembled πρόγραμμα. Οι ετικέτες αυτές πρακτικά έχουν δηλωθεί στις σειρές κώδικα που αντιστοιχούν στις συμβολικές θέσεις μνήμης 0813, 0812 και 080A, αντίστοιχα. Επισημαίνεται ότι οι διευθύνσεις άλματος γράφονται στην μνήμη με το LSByte πρώτα και το MSByte μετά. Έτσι η διεύθυνση άλματος 0812 σώζεται σαν 12 στην θέση 080C και 08 στη θέση 080D (φαίνεται αμέσως παρακάτω).

Παρακάτω δίνεται ο πίνακας που οπτικοποιεί τα όσα αναφέρθηκαν. Βρίσκεται σε πλήρη συμφωνία με τα όσα παράγει ο προσομοιωτής μLab εάν εισάγουμε τον assemby κώδικα και επιλέξουμε Πρόγραμμα --> Μνήμη και Εντολές από το κάτω μέρος του παραθύρου.

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή (Assembly)
0800	06 - opcode		MVI B, 01H
0801	01 - data		
0802	3A - opcode		LDA 2000H
0803	00 -address		
0804	20 -address		
0805	FE - opcode		CPI 00H
0806	00 - data		
0807	CA - opcode		JZ FIRST
0808	13 -address		
0809	08 -address		
080A	1F - opcode	THIRD:	RAR
080B	DA - opcode		JC SECOND
080C	12 -address		
080D	08 -address		
080E	04 - opcode		INR B
080F	C2 - opcode		JNZ THIRD
0810	0A -address		
0811	08 -address		
0812	78 - opcode	SECOND:	MOV A,B
0813	2F - opcode	FIRST:	CMA
0814	32 - opcode		STA 3000H
0815	00 -address		
0816	30 -address		
0817	CF - opcode		RST 1

Τρέχοντας, λοιπόν, το πρόγραμμά μας στο μLab παρατηρούμε πως η λειτουργία του είναι η εξής:

- αν θεωρήσουμε ότι ένα αναμμένο LED συμβολίζει το δυαδικό 1,
- ένα σβηστό LED συμβολίζει το δυαδικό 0 και
- αριθμήσουμε τους dip switches με τη σειρά από το 1 έως το 7 προς τα αριστερά

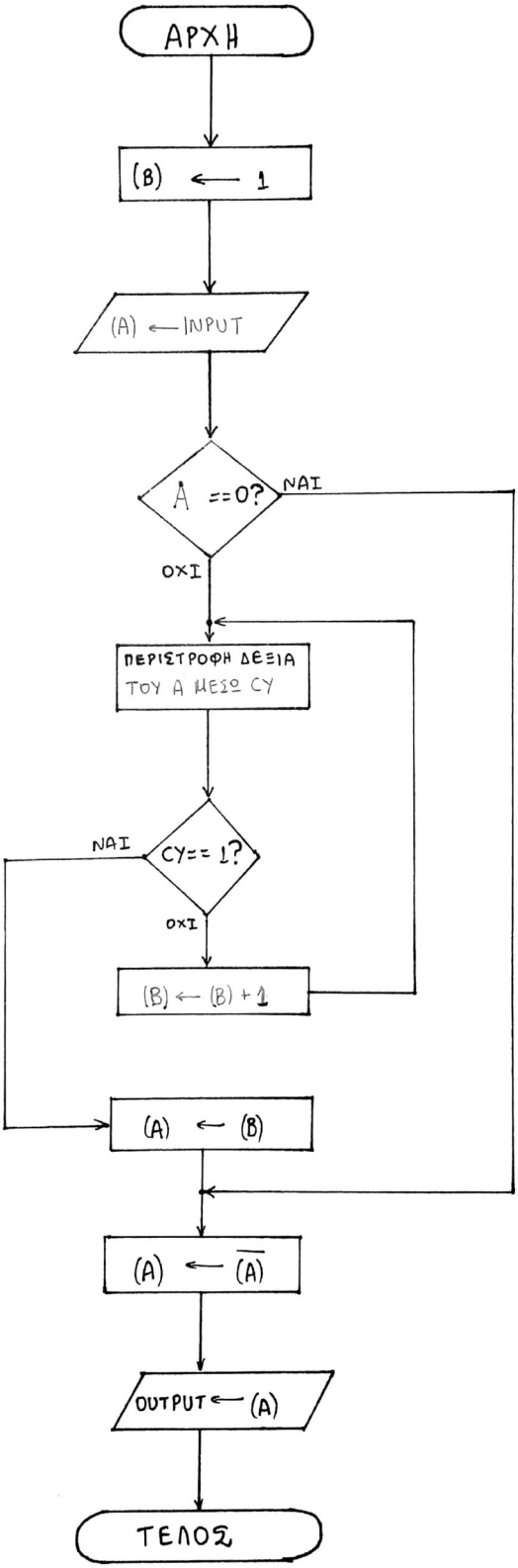
τότε τα LEDs ανάβουν με τρόπο τέτοιο ώστε ο δυαδικός αριθμός που αναπαριστούν να ταυτίζεται με τον αριθμό του δεξιότερου dip switch που βρισκόταν σε κατάσταση ON τη στιγμή που πατήθηκε το κουμπί run. Αν όλοι οι dip switches βρίσκονταν σε κατάσταση OFF, τότε δεν ανάβει κανένα LED.

Θα εξηγήσουμε, λίαν συντόμως, τον αλγόριθμο που εκτελείται προκειμένου να επιτευχθεί το αποτέλεσμα του προγράμματος:

```
        MVI B,01H       ; Ο καταχωρητής B αρχικοποιείται με 1, έτοιμος για την
                        ; περίπτωση που η είσοδος δεν είναι 0
        LDA 2000H       ; Φορτώνεται  η είσοδος των dip switches στον A
        CPI 00H         ; Σύγκριση εισόδου με το 0
        JZ FIRST        ; Αν είσοδος = 0, τότε άλμα στην εντολή CMA (καμία
                          διαφοροποίηση) και προώθηση στην έξοδο, δηλ. κανένα
                          αναμμένο LED
THIRD:  RAR             ; Περιστροφή μέσω κρατουμένου για να πάρουμε το LSB στο
                        ; flag CY
        JC SECOND       ; αν CY = 1, τότε βρήκαμε τον δεξιότερο ON dip switch
                        ; και κάνουμε άλμα στην εντολή MOV A,B για προώθηση του B
                        ; (που μετράει θέσεις μέχρι να βρούμε το πρώτο bit = 1 από
                        ; δεξιά) στον A και στην συνέχεια στην έξοδο των LEDs
        INR B           ; αύξηση του B. Θα προχωρήσουμε στο δεξιότερο bit αμέσως
        JNZ THIRD       ; άλμα στην αρχή της επαναληπτικής διαδικασίας
SECOND: MOV A,B         ; προώθηση του μετρητή B στον A ώστε να βγει στην έξοδο
FIRST:  CMA             ; αντιστροφή του A, ώστε τα LEDs (που είναι αρνητικής
                        ; λογικής) να ανάβουν για τους άσσους
        STA 3000H       ; προώθηση του A στην έξοδο. Άναμμα των αντίστοιχων LEDs
        RST 1           ; Διακοπή προγράμματος
```

Μπορούμε να επιτύχουμε τη συνεχόμενη μορφή του παραπάνω προγράμματος προσθέτοντας μια ετικέτα, έστω START, στην πρώτη γραμμή του κώδικα και μια εντολή άλματος στην ετικέτα START αμέσως πριν από την εντολή RST 1.

Παρατίθεται, στη συνέχεια, το διάγραμμα ροής του προγράμματος:



2η και 3η Άσκηση

Στον προσομοιωτή του εκπαιδευτικού προγράμματος μLab αναπτύχθηκαν τα πρόγραμματα ex2.8085 και ex3.8085 σε assembly, τέτοια ώστε να πληρούν τα ζητούμενα των αντίστοιχων εκφωνήσεων. Οι κώδικες βρίσκονται στο συμπιεσμένο αρχείο που παραδώθηκε και περιλαμβάνουν διευκρυνιστικά σχόλια αναφορικά με τις επιμέρους λειτουργίες που επιτελούν οι υπορουτίνες προκειμένου να επιτευχθεί το επιθυμητό αποτέλεσμα.

4η Άσκηση

Τεχνολογία 1: Αρχικό κόστος σχεδίασης = 20.000 €
Κόστος I.C. ανά πλακέτα = 15 €
Κόστος κατασκευής και συναρμολόγησης ανά πλακέτα = 15 €

Καμπύλη κόστους: K(x) = (15 + 15)x + 20000 ⇒ K(x) = 30x + 20000

Καμπύλη κόστους ανά τεμάχιο: $f(x)=\frac{20000}{x}+30$

Τεχνολογία 2: Αρχικό κόστος σχεδίασης = 10.000 €
Κόστος I.C. ανά πλακέτα = 60 €
Κόστος κατασκευής και συναρμολόγησης ανά πλακέτα = 10 €

Καμπύλη κόστους: K(x) = (60 + 10)x + 10000 ⇒ K(x) = 70x + 10000

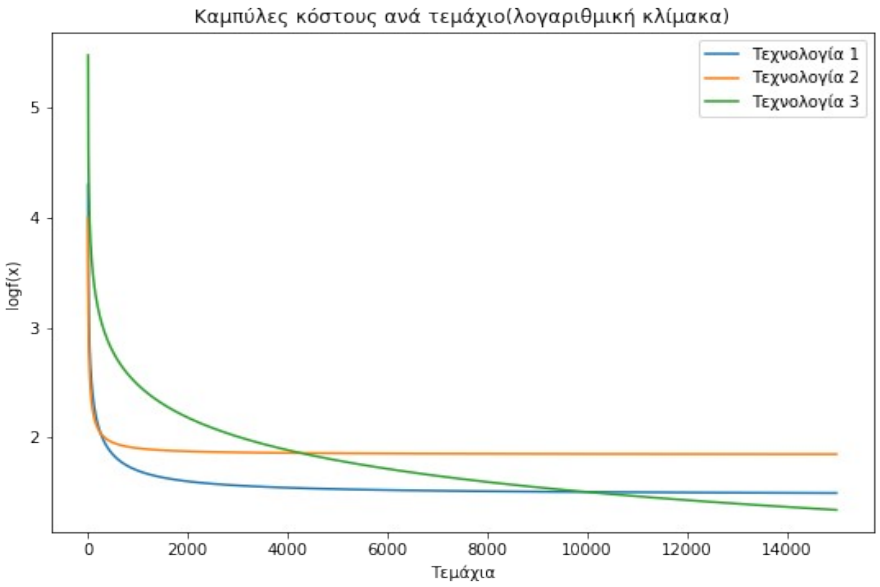
Καμπύλη κόστους ανά τεμάχιο: $f(x)=\frac{10000}{x}+70$

Τεχνολογία 3: Αρχικό κόστος σχεδίασης = 300.000 €
Κόστος I.C. ανά πλακέτα = 1 €
Κόστος κατασκευής και συναρμολόγησης ανά πλακέτα = 1 €

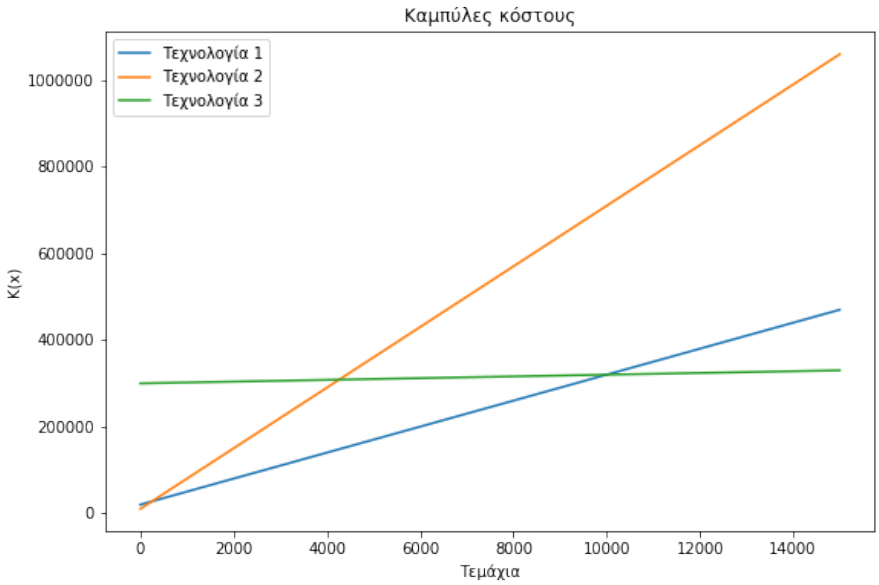
Καμπύλη κόστους: K(x) = (1 + 1)x + 300000 ⇒ K(x) = 2x + 300000

Καμπύλη κόστους ανά τεμάχιο: $f(x)=\frac{300000}{x}+2$

Βλέπουμε σε κοινό διάγραμμα τις καμπύλες κόστους ανα τεμάχιο συσκευής των τριών τεχνολογιών, σχεδιασμένες σε λογαριθμική κλίμακα, ώστε να είναι εμφανή τα σημεία τομής τους.



Βλέπουμε σε κοινό διάγραμμα τις καμπύλες κόστους των τριών τεχνολογιών.



Οι περιοχές στις οποίες συμφέρει η καθεμία τεχνολογία προκύπτουν από τα διαστήματα που ορίζουν τα σημεία τομής τους.

Περιοχή 1: Από τα διαγράμματα φαίνεται ότι αρχικά συμφέρει η τεχνολογία 2. Το πέρας της περιοχής αυτής είναι το σημείο x για το οποίο:

$$K_1(x) = K_2(x) \Rightarrow 30 \cdot x + 20000 = 70 \cdot x + 10000 \Rightarrow x = 250 \text{ τεμάχια}$$

Άρα, περιοχή 1 [0, 250): συμφέρει η τεχνολογία 2

Περιοχή 2: Από τα διαγράμματα φαίνεται ότι εδώ συμφέρει η τεχνολογία 1. Το πέρας της περιοχής αυτής είναι το σημείο x για το οποίο:

$$K_1(x) = K_3(x) \Rightarrow 30 \cdot x + 20000 = 2 \cdot x + 300000 \Rightarrow x = 10000 \text{ τεμάχια}$$

Άρα, περιοχή 2 [250, 10000): συμφέρει η τεχνολογία 1

Περιοχή 3: Από τα διαγράμματα φαίνεται ότι για μεγάλο αριθμό τεμαχίων συμφέρει η τεχνολογία 3.

Άρα, περιοχή 3 [10000, ∞): συμφέρει η τεχνολογία 3

Για να εξαφανιστεί η επιλογή της 1ης τεχνολογίας, λόγω της επιλογής των FPGAs, θα πρέπει η δεύτερη να συμφέρει έναντι της πρώτης για κάθε αριθμό τεμαχίων, ώστε η πρώτη να μην έχει νόημα και να εξαφανιστεί. Άρα:

$$K_1(x) > K_2'(x) \Rightarrow 30 \cdot x + 20000 > (10 + IC) \cdot x + 10000 \Rightarrow$$
$$30 \cdot x - (10 + IC) \cdot x + 10000 > 0 \Rightarrow 20 \cdot x - IC \cdot x + 10000 > 0 \Rightarrow$$
$$(20 - IC) \cdot x + 10000 > 0 \Rightarrow (IC - 20) \cdot x < 10000$$

Το x (που εκφράζει την ποσότητα των τεμαχίων που πωλούνται) είναι αριθμός γνήσια θετικός, άρα θα πρέπει ο συντελεστής του να είναι αρνητικός, διαφορετικά από μια τιμή του x και πάνω η ζητούμενη ανισότητα δεν θα ισχύει. Εφόσον, θέλουμε να ισχύει για κάθε δυνατή τιμή του x θα πρέπει:

$$IC - 20 < 0 \Rightarrow IC < 20$$

Τελικά, για τιμές των I.C. στην τεχνολογία των FPGAs μικρότερες των 20 € η επιλογή της 1ης τεχνολογίας εξαφανίζεται.

5η Άσκηση

- F1 = A(CD + B) + BC'D'

Περιγραφή σε επίπεδο πυλών

```
module circuit_F1 (F1, A, B, C, D);
        output F1;
        input A, B, C, D;
        wire Cnot, Dnot, w1, w2, w3, w4;

        not G1(Cnot, C),
            G3(Dnot, D);

        and G2(w1, C, D),
            G5(w3, B, Cnot, Dnot),
            G6(w4, A, w2);

        or G4(w2, B, w1),
           G7(F1, w4, w3);
endmodule
```

Μοντελοποίηση Ροής Δεδομένων

```
module circuit_F1 (F1, A, B, C, D);
        output F1;
        input A, B, C, D;
        assign F1 = (A & ((C & D) || B)) | (B & ~C & ~D);
endmodule
```

- F2(A, B, C, D) = Σ(0,2,3,5,7,8,10,11,14,15)

Περιγραφή σε επίπεδο πυλών

```
primitive UDP_F2 (F2, A, B, C, D);
        output F2;
        input A, B, C, D;
        table          // πίνακας αληθείας για την F2
 //     A       B       C       D:      F2      // column header
        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:      1;
        1       0       0       1:      0;
        1       0       1       0:      1;
        1       0       1       1:      1;
        1       1       0       0:      0;
        1       1       0       1:      0;
        1       1       1       0:      1;
        1       1       1       1:      1;
        endtable
endprimitive
```

Μοντελοποίηση Ροής Δεδομένων

```
primitive UDP_F2 (F2, A, B, C, D);
        output F2;
        input A, B, C, D;
        assign 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);
endprimitive
```

- F3 = ABC + (A + B)CD + (B + CD)E

Περιγραφή σε επίπεδο πυλών

```
module circuit_F3 (F3, A, B , C, D, E);
        output F3;
        input A, B, C, D, E;
        wire w1, w2, w3, w4, w5, w6;

        and G1(w1, C, D),
              G4(w4, A, B, C),
              G5(w5, w2, C, D),
              G6(w6, w3, E);

        or G2(w2, A, B),
           G3(w3, B, w1),
           G7 (F3, w4,w5,w6);
endmodule
```

Μοντελοποίηση Ροής Δεδομένων

```
module circuit_F3 (F3, A, B , C, D, E);
        output F3;
        input A, B, C, D, E;
        assign F3 = (A & B & C) | ((A | B) & C & D) | ((B | (C & D)) & E);
endmodule
```

- F4 = A(BC + D + E) + CDE

Περιγραφή σε επίπεδο πυλών

```
module circuit_F4 (F4, A, B, C, D, E);
        output F4;
        input A, B, C, D, E;
        wire w1, w2, w3, w4;

        and G1(w1, B, C),
              G2(w2, C, D, E),
              G4(w4, A, w3);

        or G3(w3, w1, D, E),
           G5(F4, w4, w2);
endmodule
```

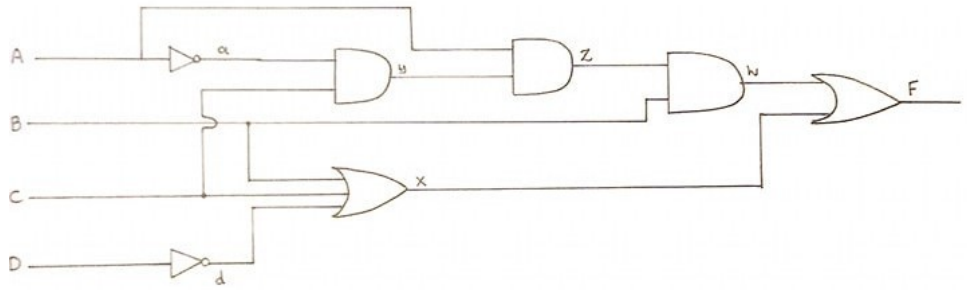
Μοντελοποίηση Ροής Δεδομένων

```
module circuit_F4 (F4, A, B, C, D, E);
        output F4;
        input A, B, C, D, E;
        assign F4 = (A & ((B & C) | D | E)) | (C & D & E);
endmodule
```

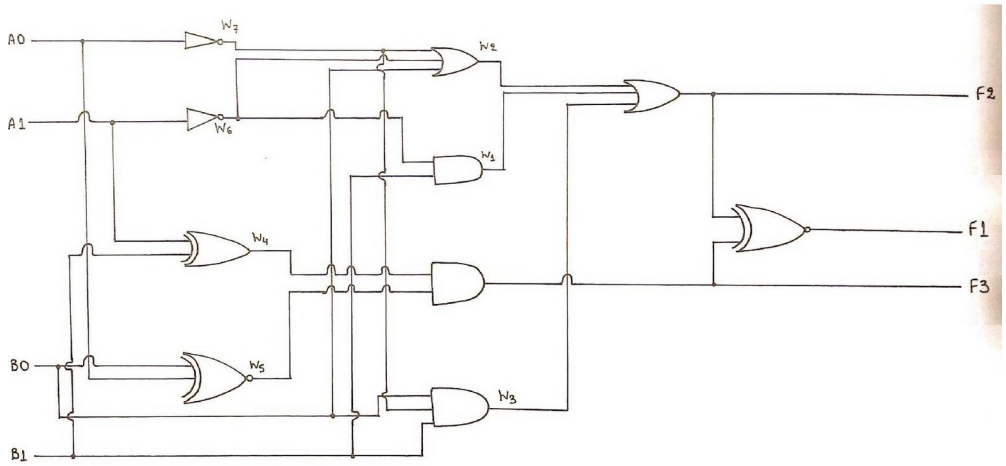
6η Άσκηση

(i)

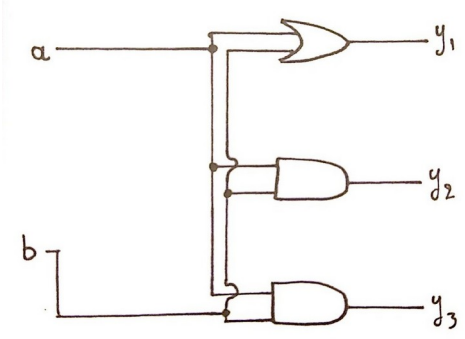
(a)



(b)



(c)



(ii)

```
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);
        half_adder HA2(S, C2, C1, z);
        or G1(C, C2, C1);
endmodule

module _4_bit_add_sub(output [3:0] S, output C, V,
                              input [3:0] A, B, input M);
        wire C1, C2, C3;
        wire w0, w1, w2, w3;

        xor G1(w0, B[0], M);
        xor G2(w1, B[1], M);
        xor G3(w2, B[2], M);
        xor G4(w3, B[3], M);

        full_adder FA0 (S[0], C1, w0, A[0], M);
        full_adder FA1 (S[1], C2, w1, A[1], C1);
        full_adder FA2 (S[2], C3, w2, A[2], C2);
        full_adder FA3 (S[3], C, w3, A[3], C3);

        xor G5(V, C, C3);
endmodule
```

(iii)

```
module _4_bit_add_sub(output [3:0] S, output C, V,
                              input [3:0] A, B, input M);
        assign {C, S} = M ? A – B : A + B;
endmodule
```

7η Άσκηση

(i) Μηχανή Mealy

```
module Mealy_Machine (
        output reg y_out,
        input x_in, clock, reset
);
        reg [1:0]       state, next_state;
        parameter       a = 2'b00,
                        b = 2'b11,
                        c = 2'b10,
                        d = 2'b01;
        always @ (posedge clock, negedge reset)     // state update or reset
                if (reset == 0) state <= a;
                else state <= next_state;
        always @ (state, x_in)
                case (state)
                        a: if (x_in) next_state = c; else next_state = b;
                        b: if (x_in) next_state = d; else next_state = c;
                        c: if (x_in) next_state = d; else next_state = b;
                        d: if (x_in) next_state = a; else next_state = c;
                endcase
        always @ (state, x_in)      // output formation
                case (state)
                        a, d: y_out = ~x_in;
                        b, c: y_out = x_in;
                endcase
endmodule
```

(ii) Μηχανή Moore

```
module Moore_Machine (
        output y_out,
        input x_in, 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_in) state <= c; else state <= b;
                        b: if (x_in) state <= d; else state <= c;
                        c: if (x_in) state <= d; else state <= b;
                        d: if (x_in) state <= a; else state <= c;
                endcase
        always @ (state)
                case (state)
                        a, d: y_out <= 1'b0;
                        b, c: y_out <= 1'b1;
                endcase
endmodule
```

(ii) 4-bit άνω και κάτω μετρητής 2 εισόδων (Up/Down και Clear) σύγχρονων με το ρολόι

```
module up_down_counter (
        output reg [3:0] count,
        input clk, mode, clear
);
        always @ (posedge clk, negedge clear)
                if (clear == 0) count <= 4'b0000;
                else case (mode)
                        1'b1: count <= count + 4'b0001;         // up counting
                        1'b0: count <= count – 4'b0001;         // down counting
                        default: count <= count;                // πλεονάζουσα δήλωση
                endcase
endmodule
```