Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Συστήματα Μικροϋπολογιστών, ροή Υ, 6^{ο} εξάμηνο

Αναφορά 1^{ης} Σειράς Ασκήσεων,

Στοιχεία σπουδαστών

Όνομα: Μαντζούτας Ανδρέας Α.Μ. : 03117108

Όνομα: Τσιτσής Αντώνιος Α.Μ. : 03117045

Εξάμηνο: 6^{ο}

Ημερομηνία: 29/04/2020

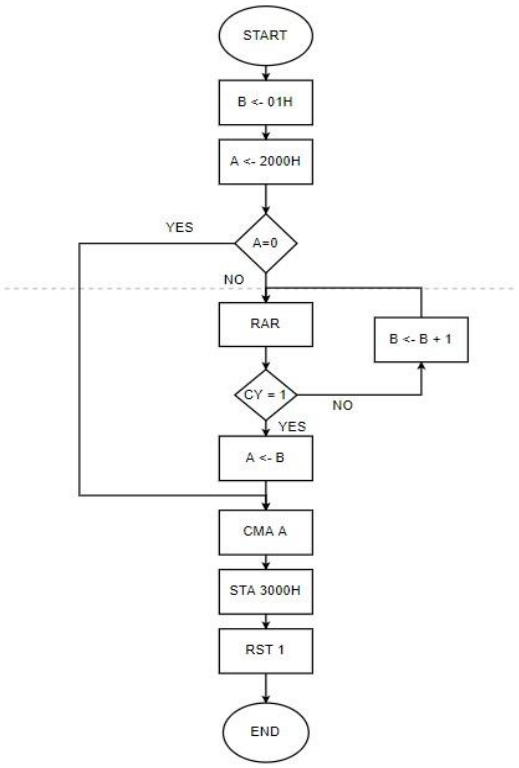
1η Άσκηση

Με τη βοήθεια του πίνακα 2 του παραρτήματος 2, συντάξαμε τον κώδικα assembly που ζητήθηκε, καθώς και το διάγραμμα ροής του προγράμματος:

```
START:
        MVI B,01H
        LDA 2000H
        CPI 00H
        JZ ADR3
ADRA:
        RAR
        JC ADR2
        INR B
        JNZ ADRA

ADR2:
        MOV A,B

ADR3:
        CMA
        STA 3000H
        RST 1
END
```



Έπειτα, για την επίτευξη μιας συνεχόμενης ροής στη λειτουργία του προγράμματος, αρκεί να βάλουμε ένα «άλμα» πριν το END που να οδηγεί το πρόγραμμα στην ετικέτα START. Αυτό μπορεί να επιτευχθεί με χρήση της εντολής :

JMP START

Ο κώδικας επισυνάπτεται με όνομα ask1.8085

2η Άσκηση

```
             IN 10H
             LXI B,0244H ; KA8ISTERISH 1/2second
             MVI D,01H ;arxikopoisi D
             CMA
             STA 3000H


DECISION:    CALL DELB ;klisi kathisterisis
             LDA 2000H
             ANI 02H ; an to 2o lsb einai on, freeze
             CPI 00H
             JZ FREEZE
             LDA 2000H
             ANI 01H  ;mask 1st lsb
             CPI 00H
             JNZ PERADOTHE
             JMP NORMALMOVE

NORMALMOVE:  MVI E,00H ;apli kukliki kinisi 123456781234...
             MOV A,D
             RLC
             MOV D,A
             CMA
             STA 3000H
              JMP DECISION

PERADOTHE:    MOV A,E ;kinisi 123456787654...
             CPI 00H   ;an a!=0, pigaine anapoda
             JNZ DOTHE
             JMP PERA


PERA:        MVI E,00H
             MOV A,D
             CPI 80H  ;an eimai sto teleutaio led, pao deksia
             JNC DOTHE
             RLC    ; kinoume aristera
             MOV D,A
             CMA
             STA 3000H
             JMP DECISION

DOTHE:        MOV A,D
             MVI E,01H
             CPI 01H ;an eimai sto proto led, pao aristera
             JZ PERA
             RRC      ;kinoume deksia
             MOV D,A
             CMA
             STA 3000H
             JMP DECISION


FREEZE:
       MVI A,FEH
       STA 3000H
       JMP DECISION


END
```

Ο κώδικας επισυνάπτεται με όνομα ask2.808

3η Άσκηση

```
START:
       MVI D,00H
       LDA 2000H ;diasma eisodou
       CPI 64H ;sygkrisi me 100
       JNC MODULO ;an megalutero 100, pigaine sto modulo
       JMP CORE

MODULO:

       SUI 64H ;afairese 100
       CPI 64H ;an megalitero 100, ksana sto modulo
       JNZ MODULO
       JMP CORE

CORE:
       SUI 0AH ;afairese10
       INR D ;metra tis 10ades
       CPI 0AH ;an megalutero toy 10
       JNZ CORE ;ksana afairese 10
       MOV C,A ;store to periexomeno tou A ston c
       MOV A,D ; store to D ston B
       RLC ; 4 rotate oste na ftiaksoume ta 4 msb
       RLC
       RLC
       RLC
       ADD C ; prosthese ta 4 lsb
       CMA
       STA 3000H ;ektupose
       RST 1
       JMP START
END
```

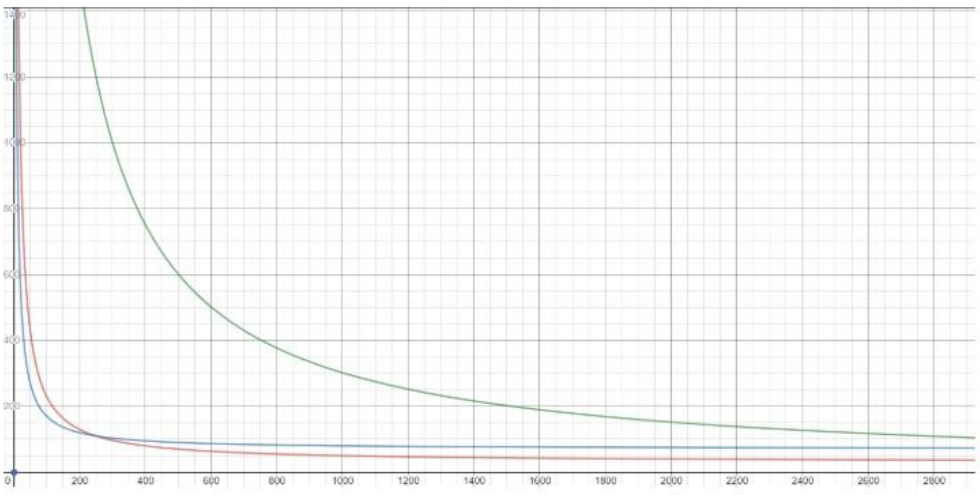
Ο κώδικας επισυνάπτεται με όνομα ask3.8085

4η Άσκηση

Με τις δοθείσες τεχνολογίες οι συναρτήσεις κόστους-αριθμου τεμαχίων είναι οι εξής:

1. Κόστος = 20.000 /τεμάχια + (15 + 15) (I.C.)
2. Κόστος = 10.000 /τεμάχια + (60 + 10) (FPGAs)
3. Κόστος = 600.000/τεμάχια + (1 + 1) (SoC)

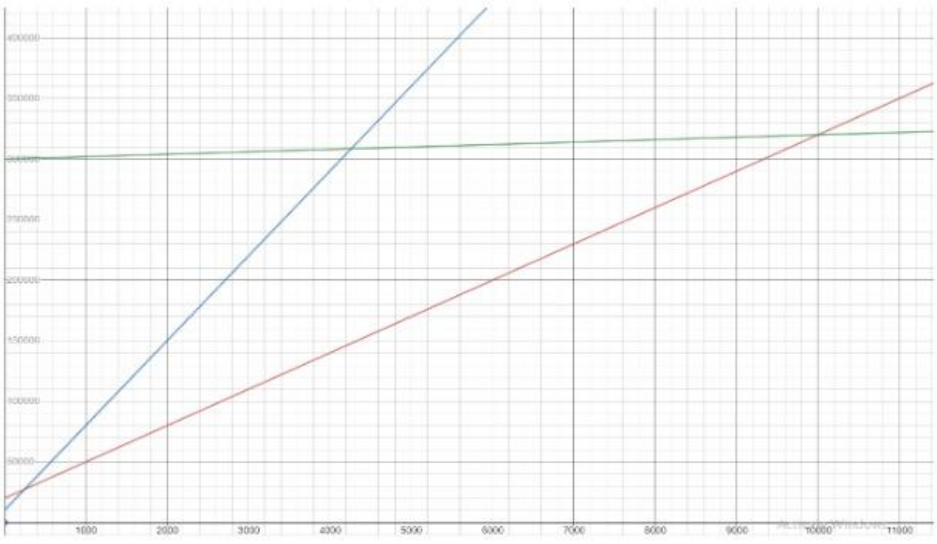
Και οι γραφικές τους :



Τα συνολικά κόστη είναι:

1. Κόστος = 20.000 + (15 + 15)*τεμάχια (I.C.)
2. Κόστος = 10.000 + (60 + 10)*τεμάχια (FPGAs)
3. Κόστος = 600.000 + (1 + 1)*τεμάχια (SoC)

Και οι γραφικές τους :

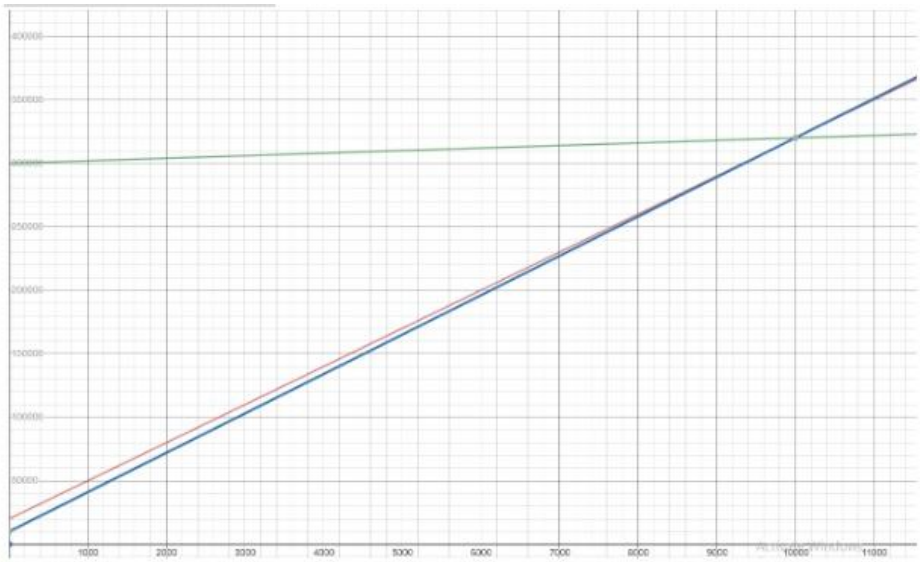


Συνεπώς φαίνεται ότι ανάλογα με το πόσα τεμάχια θέλουμε να παράγουμε επιλέγουμε τις εξής τεχνολογίες:

- Από 0 – 250 τεμάχια την FPGAs
- Από 250 – 10000 τεμάχια την I.C.
- Από 10000 - άπειρα τεμάχια την SoC

Για να εξαφανιστεί η επιλογή 1(I.C.) αλλάζοντας το κόστος παραγωγής της τεχνολογίας FPGAs πρέπει το κόστος αυτό να γίνει μικρότερο ή ίσο του 20,5 € (ώστε οι καμπύλες κόστους των τριών επιλογών να τέμνονται στο ίδιο σημείο)

Άρα έχω την γραφική :



Και άρα οι συμφέρουσες επιλογές αναλόγως το πλήθος των τεμαχίων γίνονται:

- Από 0 – 10000 τεμάχια την FPGAs
- Από 10000 - άπειρα τεμάχια την SoC

5η Άσκηση

(i)Δομική Περιγραφή

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

```
module F1 (E, A, B, C ,D);
        input A, B, C ,D;
        output E;
        wire notC, notD, w1, w2, w3, w4;
        not  G1(notC, C);
        not G2(notD, D);
        and G3(w1, B, notC, notD);
        and G4 (w2, C, D);
        and G5 (w3, w4, A);
        or G6 (w4, w2, B);
        or G7(E, w3, w1);
endmodule
```

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

```
primitive F2_help(e, a, b, c, d);
output e;
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 : 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


module F2(E, A, B, C, D);
        input A, B, C, D;
        output E;
        F2_help(E, A, B, C, D);
Endmodule
```

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

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

        and G1(w1, A, B, C);
        and G2(w2, C, D, w5);
        and G3(w3, C, D);
        and G4(w4, w6, E);

        or G5(w5, A, B);
        or G6(w6, B, w3);
        or G7(F, w1, w2, w4);
endmodule
```

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

```
module circuit_with_F4 (F, A, B, C, D, E);
        output F;
        input A, B, C, D, E;
        wire w1, w2, w3, w4;

        and G1(w1, B, C);
        and G2(w2 C, D, E);
        and G3(w3, w4, A);
        or G4(w4, w1, D, E);
        or G5(F, w3, w2);
endmodule
```

(ii) Μοντελοποίηση Ροής Δεδομένων

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

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

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

```
module F2(E, A, B, C ,D);
        output E;
        input A, B, C, D;
        assign E = ((~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));
endmodule
```

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

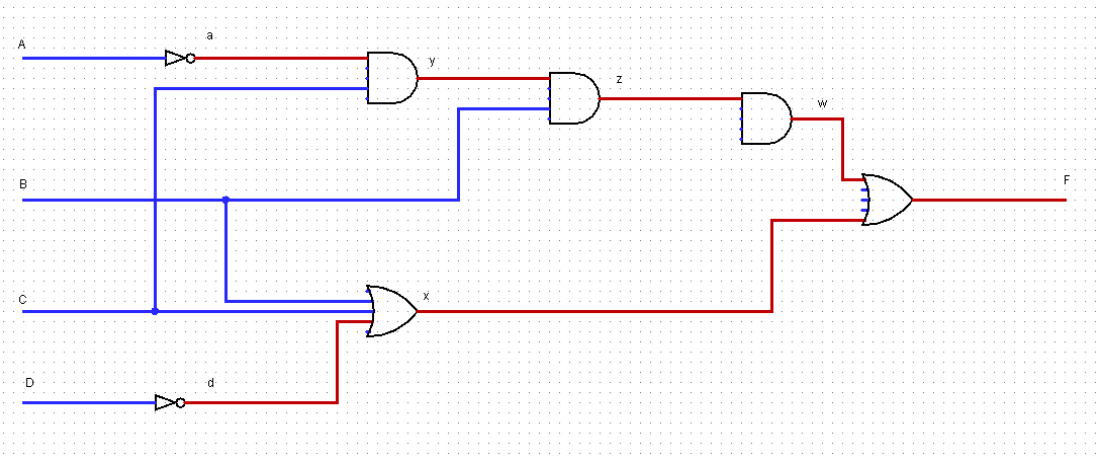
```
module F3(F, A, B, C, D, E);
        output F;
        input A, B, C, D, E;
        assign F = ((A & B & C) | ((A | B) & C & D) | (( B | (C & D)) & E);
endmodule
```

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

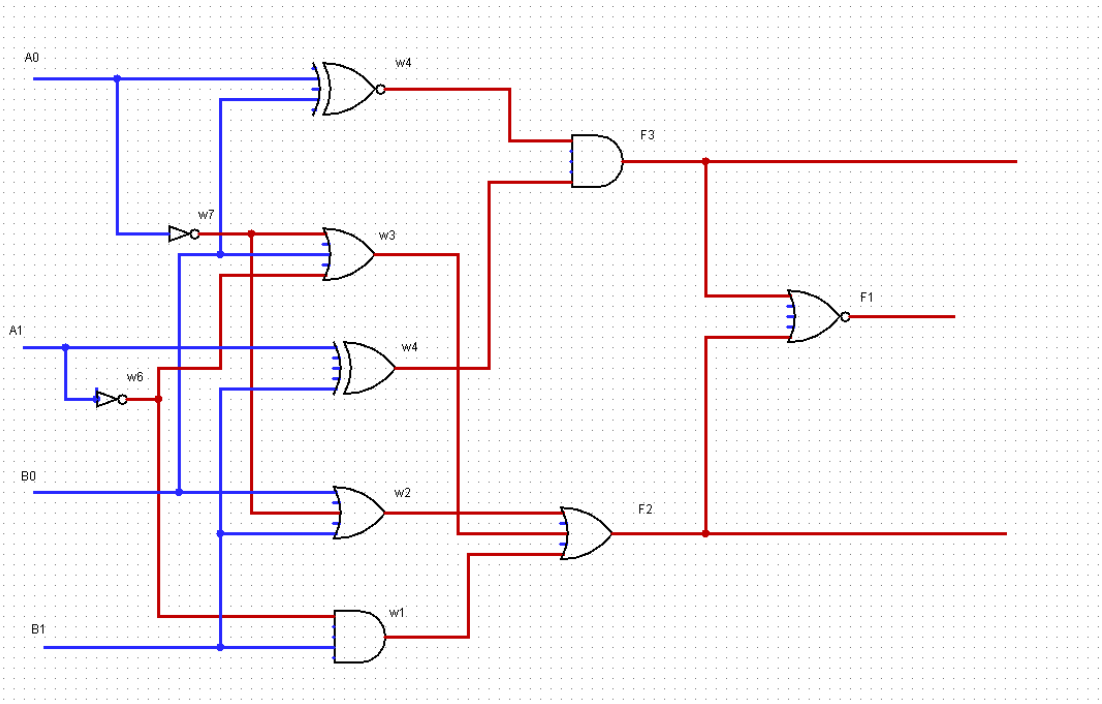
```
module F4(F, A, B, C, D, E);
        output F;
        input A, B, C, D, E;
        assign F = ((A & ( (B & C) | D | E)) | ( C & D & E));
endmodule
```

6η Άσκηση

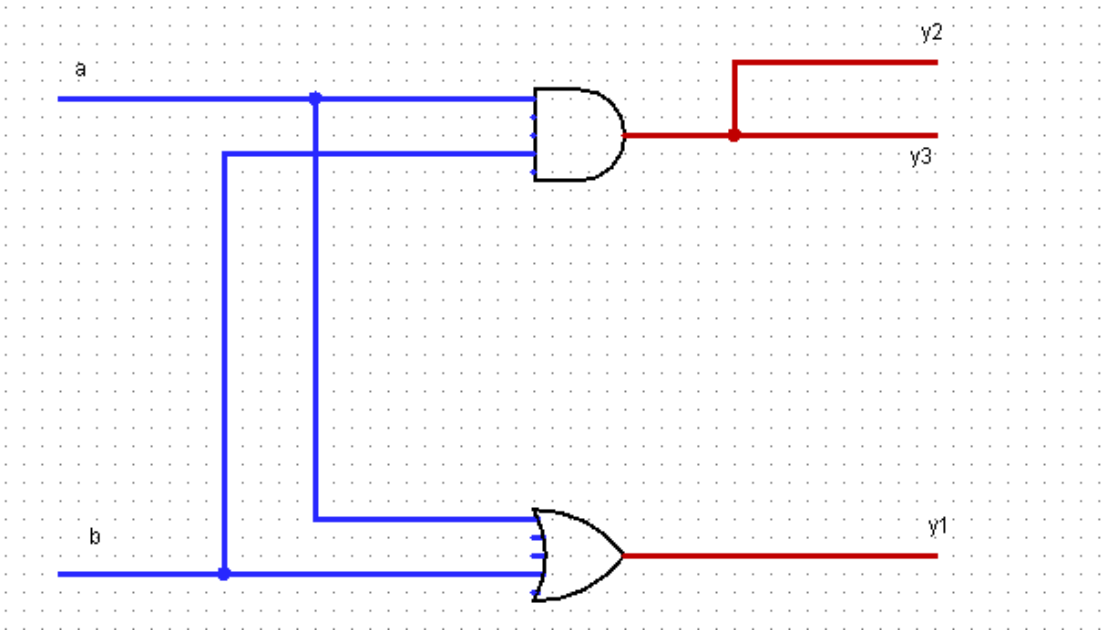
α)



β)



γ)



(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 w1,w2,c2;
        half_adder G1(w1,w2,x,y);
        half_adder G2(S,w3,w1,z);
        or G3(C,w2,w1);
endmodule

module 4_bit_adder_subtractor(output [3:0] Sum , output C0 , input [3:0]A , input
[3:0] B,input M);
        wire c1,c2,c3,B0,B1,B2,B3;
        xor
                G0(B0,B[0],M),
                G1(B1,B[1],M),
                G2(B2,B[2],M),
                G3(B3,B[3],M);
        full_adder
                FA0(Sum[0],c1,A[0],B0,M),
                FA1(Sum[1],c2,A[1],B[1],M),
                FA2(Sum[2],c2,A[2],B[2],M),
                FA3(Sum[3],c3,A[3],B[3],M);
endmodule
```

(iii)

```
module 4bit_Adder_Subtractor (Sum,Cout,A,B,select);
        output [3:0]Sum ;
        output Cout;
        input [3:0] A, B;
        input select;
        assign { Cout,Sum } = select ? (A – B) : (A + B);
endmodule
```

7η Άσκηση

(i) Mealy FSM

```
module Mealy_fsm(y, x, clock, reset);
        output reg 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 <= c; else state <=b;
                b:  if (x) state <= d; else state <=c;
                c:  if (x) state <= d; else state <=b;
                d:  if (x) state <= a; else state <=c;
        endcase
        assign y_out = ((~state[0]  & state[1] & x) | (state[0] & ~state[1] & x) |
                        (state[0] & state[1] & ~x) | ( ~state[0] & ~state[1] & ~x));
endmodule
```

(ii) Moore Fsm

```
module Moore_fsml(x,y,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<=c else state<=b;
                b: if (x) state<=d else state<=c;
                c: if (x) state<=d else state<=b;
                d: if (x) state<=a else state<=c;
        endcase
        assign y = state[1] ^ state[0];

endmodule
```

(iii)

```
module up_down4bitcounter(up_down, clear, CLK, y_out);
        input up_down, CLK, clear;
        output y_out;
        reg [0:3]A;
        assing y_out = ~up_down & (A == 4'b1111);
        always @ (possedge CLK, negedge clear)
        if (~clear) A <= 4'b0000;
        else if (up_down) A <= A – 1b'1;
        else A<= A + 1b'1;
endmodule
```