

# HARDWIRED

## ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

4

ΕΠΩΝΥΜΟ: ΣΤΑΜΑΤΑΚΗΣ

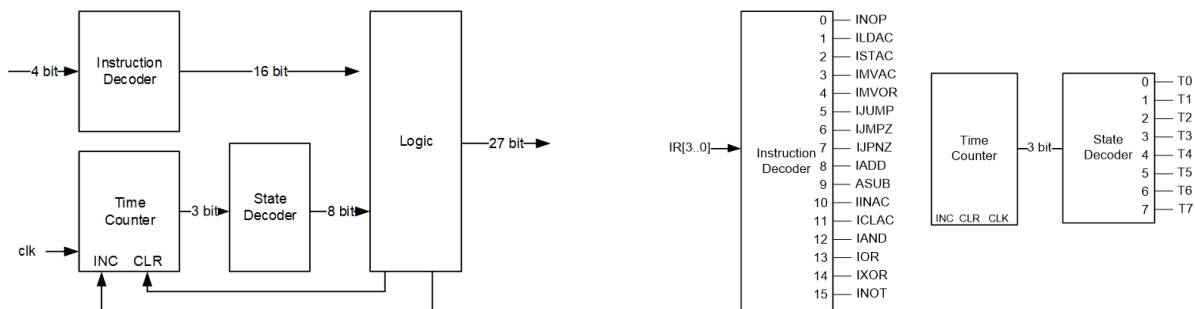
ΟΝΟΜΑ : ΦΩΤΗΣ

ΑΜ: 21048

### Σκοπός

Με αφορμή την σχεδίαση και την εξομοίωση με διάφορους τρόπους, απλών ψηφιακών κυκλωμάτων θα κατακτηθεί το αντικείμενο της άσκησης αυτής που είναι η σχεδίαση της μονάδας ελέγχου, χρησιμοποιώντας την hardwired λογική η οποία θα χρησιμοποιηθεί, εναλλακτικά με την microprogrammed, κατά την τελική σύνθεση της σχετικά απλής ΚΜΕ.

Η μονάδα ελέγχου είναι αυτή που παρέχει στην ΚΜΕ τα απαραίτητα σήματα ελέγχου για τη λειτουργία της. Η λογική σχεδίασής της θα είναι η hardwired λογική, η οποία θα υλοποιηθεί με μία μηχανή πεπερασμένων καταστάσεων – FSM. Η μηχανή καταστάσεων αποτελείται από δύο αποκωδικοποιητές, ένα μετρητή και ένα συνδυαστικό κύκλωμα. Ο πρώτος αποκωδικοποιητής (instruction decoder) παράγει ένα ξεχωριστό σήμα για κάθε εντολή ενώ ο δεύτερος αποκωδικοποιητής (state decoder), με τη βοήθεια ενός απαριθμητή (time counter), παρακολουθεί ποια κατάσταση του κύκλου ανάκλησης η εκτέλεσης κάθε εντολής είναι ενεργή. Τέλος μια μονάδα συνδυαστικής λογικής παράγει μέσα από τα ξεχωριστά σήματα, σήματα ελέγχου για κάθε αποκωδικοποιητή αλλά και για τον απαριθμητή. Μια τέτοια μονάδα ελέγχου θα είχε την ακόλουθη μορφή (Σχήμα 1a).



(a) (b)  
Σχήμα 1: Λογικό διάγραμμα HardWired Μονάδας Ελέγχου.

Η σχεδίαση του αποκωδικοποιητή εντολών είναι σχετικά απλή. Δέχεται σαν είσοδο την έξοδο του καταχωρητή εντολών (IR) ενώ δεδομένου ότι χρησιμοποιούνται μόνο τα 4 bit του καταχωρητή εντολών για το ρεπερτόριο των 16 εντολών της σχετικά απλής ΚΜΕ είναι προφανές ότι ο αποκωδικοποιητής εντολών είναι ένα αποκωδικοποιητής 4 σε 16. Από την άλλη εφόσον ο μέγιστος αριθμός καταστάσεων για το ρεπερτόριο των 16 εντολών είναι 8 καταστάσεις στη σχεδίαση μας χρησιμοποιούμε έναν απαριθμητή 3 bit με δυνατότητα αύξησης και μηδενισμού και ένα αποκωδικοποιητή 3 σε 8. Τα παραπάνω στοιχεία και οι έξοδοι τους φαίνονται με μεγαλύτερη λεπτομέρεια στο Σχήμα 1b.

Η ρουτίνα FETCH είναι η μόνη ρουτίνα η οποία δεν χρησιμοποιείται από το αποκωδικοποιητή εντολών. Δεδομένου ότι κατά τη ρουτίνα αυτή η προς εκτέλεση εντολή ανακαλείται από τη μνήμη η έξοδος του αποκωδικοποιητή μπορεί να είναι οποιαδήποτε. Σε αυτή μας τη σχεδίαση αναθέτουμε την κατάσταση T0 στην FETCH1 θέλοντας να εκμεταλλευτούμε το γεγονός ότι αυτή είναι προσπελάσιμη καθαρίζοντας (clear) τον απαριθμητή καταστάσεων. Όμοια αναθέτουμε την κατάσταση T1 και T2 στην FETCH2 και FETCH3 αντίστοιχα. Οι καταστάσεις των προς εκτέλεση εντολών εξαρτώνται αφενός από το opcode κάθε εντολής και αφετέρου από την τιμή του απαριθμητή καταστάσεων. Η T3 είναι η πρώτη χρονικά κατάσταση κάθε εντολής, η T4 η δεύτερη και ούτω καθεξής. Η μονάδα ελέγχου συνδέοντας με λογική and την κατάλληλη τιμή του απαριθμητή καταστάσεων με την έξοδο του αποκωδικοποιητή εντολών παράγει τις επιμέρους καταστάσεις για κάθε εντολή. Για παράδειγμα οι δύο πρώτες καταστάσεις της εντολής LDAC είναι:

$$LDAC1 = ILDAC \wedge T3$$

$$LDAC2 = ILDAC \wedge T4$$

Η συνολική λίστα των επιμέρους καταστάσεων για όλες τις εντολές δίνεται στο πίνακα Γ.4.1 που ακολουθεί.

κατάσταση	λειτουργία	κατάσταση	λειτουργία
<b>FETCH1</b>	T0	<b>JMPZY1</b>	IJMPZ $\wedge$ Z $\wedge$ T3
<b>FETCH2</b>	T1	<b>JMPZY2</b>	IJMPZ $\wedge$ Z $\wedge$ T4
<b>FETCH3</b>	T3	<b>JMPZY3</b>	IJMPZ $\wedge$ Z $\wedge$ T5
<b>NOP1</b>	INOP $\wedge$ T3	<b>JMPZN1</b>	IJMPZ $\wedge$ Z' $\wedge$ T3
<b>LDAC1</b>	ILDAC $\wedge$ T3	<b>JMPZN2</b>	IJMPZ $\wedge$ Z' $\wedge$ T4
<b>LDAC2</b>	ILDAC $\wedge$ T4	<b>JPNZY1</b>	IJPNZ $\wedge$ Z' $\wedge$ T3
<b>LDAC3</b>	ILDAC $\wedge$ T5	<b>JPNZY2</b>	IJPNZ $\wedge$ Z' $\wedge$ T4
<b>LDAC4</b>	ILDAC $\wedge$ T6	<b>JPNZY3</b>	IJPNZ $\wedge$ Z' $\wedge$ T5
<b>LDAC5</b>	ILDAC $\wedge$ T7	<b>JPNZN1</b>	IJPNZ $\wedge$ Z $\wedge$ T3
<b>STAC1</b>	ISTAC $\wedge$ T3	<b>JPNZN2</b>	IJPNZ $\wedge$ Z $\wedge$ T4
<b>STAC2</b>	ISTAC $\wedge$ T4	<b>ADD1</b>	IADD $\wedge$ T3
<b>STAC3</b>	ISTAC $\wedge$ T5	<b>SUB1</b>	ISUB $\wedge$ T3
<b>STAC4</b>	ISTAC $\wedge$ T6	<b>INAC1</b>	IINAC $\wedge$ T3
<b>STAC5</b>	ISTAC $\wedge$ T7	<b>CLAC1</b>	ICLAC $\wedge$ T3
<b>MVAC1</b>	IMVAC $\wedge$ T3	<b>AND1</b>	IAND $\wedge$ T3
<b>MOVR1</b>	IMOVR $\wedge$ T3	<b>OR1</b>	IOR $\wedge$ T3
<b>JUMP1</b>	IJUMP $\wedge$ T3	<b>XOR1</b>	IXOR $\wedge$ T3
<b>JUMP2</b>	IJUMP $\wedge$ T4	<b>NOT1</b>	INOT $\wedge$ T3
<b>JUMP3</b>	IJUMP $\wedge$ T5		

**Πίνακας 1:** Παραγωγή καταστάσεων για τη σχετικά απλή ΚΜΕ

Έχοντας δημιουργήσει τις επιμέρους καταστάσεις για κάθε εντολή είναι ανάγκη να δημιουργήσουμε τα σήματα που θα οδηγούν τις εισόδους inc και clr του απαριθμητή καταστάσεων. Για να το

Σήμα	Συνδιαστική Λογική
<b>ARLOAD</b>	FETCH1∨FETCH3∨LDAC3∨STAC3
<b>ARINC</b>	LDAC1∨STAC1∨JMPZY1∨JPNZY1
<b>PCLOAD</b>	JUMP3∨JMPZY3∨JPNZY3
<b>PCINC</b>	FETCH2∨LDAC1∨LDAC2∨STAC1∨STAC2∨JMPZN1∨JMPZN2∨JPNZN1∨JPNZN2
<b>DRLOAD</b>	FETCH2∨LDAC1∨LDAC2∨LDAC4∨STAC1∨STAC2∨STAC4∨JUMP1∨JUMP2∨JMPZY1∨JMPZY2∨JPNZY1∨JPNZY2
<b>TRLOAD</b>	LDAC2 ∨STAC2 ∨JUMP2 ∨JMPZY2 ∨JPNZY2
<b>IRLOAD</b>	FETCH3
<b>RLOAD</b>	MVAC1
<b>ACLOAD</b>	LDAC5∨MOVR1∨ADD1∨SUB1∨INAC1∨CLAC1∨AND1∨OR1∨XOR1∨NOT1
<b>ZLOAD</b>	LDAC5∨MOVR1∨ADD1∨SUB1∨INAC1∨CLAC1∨AND1∨OR1∨XOR1∨NOT1
<b>READ</b>	FETCH2∨LDAC1∨LDAC2∨LDAC4∨STAC1∨STAC2∨JUMP1∨JUMP2∨JMPZY1∨JMPZY2∨JPNZY1∨JPNZY2
<b>WRITE</b>	STAC5
<b>MEMBUS</b>	FETCH2∨LDAC1∨LDAC2∨LDAC4∨STAC1∨STAC2∨JUMP1∨JUMP2∨JMPZY1∨JMPZY2∨JPNZY1∨JPNZY2
<b>BUSMEM</b>	STAC5
<b>PCBUS</b>	FETCH1 or FETCH3
<b>DRBUS</b>	LDAC2∨LDAC3∨LDAC5∨STAC2∨STAC3∨STAC5∨JUMP2∨JUMP3∨JMPZY2∨JMPZY3∨JPNZY2∨JPNZY3
<b>TRBUS</b>	LDAC3∨STAC3∨JUMP3∨JMPZY3∨JPNZY3
<b>RBUS</b>	MOVR1∨ADD1∨SUB1∨AND1∨OR1∨XOR1
<b>ACBUS</b>	STAC4∨MVAC1
<b>ANDOP</b>	AND1
<b>OROP</b>	OR1
<b>XOROP</b>	XOR1
<b>NOTOP</b>	NOT1
<b>ACINC</b>	INAC1
<b>ACZERO</b>	CLAC1
<b>PLUS</b>	ADD1
<b>MINUS</b>	SUB1

επιτύχουμε αυτό συνδέουμε με λογική or την τελευταία κατάσταση κάθε εντολής για να δημιουργήσουμε το σήμα που θα οδηγήσει την είσοδο clr. Δεδομένου ότι η είσοδος inc πρέπει να είναι ενεργοποιημένη σε κάθε άλλη κατάσταση , μπορεί να υλοποιηθεί συνδέοντας με λογική or όλες τις υπόλοιπες καταστάσεις (πλην της τελευταίας) κάθε εντολής. Τέλος, η συνδιαστική λογική που χρειάζεται για να παραχθούν τα κατάλληλα σήματα ελέγχου , για τα επιμέρους τμήματα της ΚΜΕ φαίνονται στο Πίνακα 2 που ακολουθεί:

**Πίνακας 2:** Παραγωγή σημάτων ελέγχου για τη σχετικά απλή ΚΜΕ

## Αποκωδικοποιητής Εντολών

Γράψτε τον κώδικα για τον αποκωδικοποιητή 4 σε 16 με σήμα εισόδου  $D_{in}$  εύρους 4 bit και σήμα εξόδου  $D_{out}$  εύρους 16 bit. Το κύκλωμα αυτό όπως είναι γνωστό θα αντιστοιχεί την τιμή (opcode) κάθε μιας από τις 16 εντολές που εμφανίζεται στην είσοδο του σε μία από τις 16 εξόδους του.

[Γράψτε εδώ το πρόγραμμά σας:](#)

**Πρόγραμμα 1:** Ο αποκωδικοποιητής εντολών.

--FOTIS STAMATAKIS--

library ieee;

use ieee.std\_logic\_1164.all;

entity instruction\_decoder is

port(

    Din : in std\_logic\_vector(3 downto 0);

    Dout : out std\_logic\_vector(15 downto 0)

);

end instruction\_decoder;

architecture rtl of instruction\_decoder is

begin

    process(Din)

    begin

        Dout <= (others => '0');

        case Din is

            when "0000" => Dout(0) <= '1'; -- NOP

            when "0001" => Dout(1) <= '1'; -- LDAC

```

when "0010" => Dout(2) <= '1'; -- STAC
when "0011" => Dout(3) <= '1'; -- MVAC
when "0100" => Dout(4) <= '1'; -- MOVR
when "0101" => Dout(5) <= '1'; -- JUMP
when "0110" => Dout(6) <= '1'; -- JMPZ
when "0111" => Dout(7) <= '1'; -- JPNZ
when "1000" => Dout(8) <= '1'; -- ADD
when "1001" => Dout(9) <= '1'; -- ASUB
when "1010" => Dout(10) <= '1'; -- INAC
when "1011" => Dout(11) <= '1'; -- CALC
when "1100" => Dout(12) <= '1'; -- AND
when "1101" => Dout(13) <= '1'; -- OR
when "1110" => Dout(14) <= '1'; -- XOR
when "1111" => Dout(15) <= '1'; -- NOT
when others => null;

end case;

end process;

end rtl;

```

## **Αποκωδικοποιητής Καταστάσεων**

Γράψτε τον κώδικα για τον αποκωδικοποιητή 3 σε 8 με σήμα εισόδου  $D_{in}$  εύρους 3 bit και σήμα εξόδου  $D_{out}$  εύρους 8 bit. Το κύκλωμα αυτό θα αντιστοιχεί την τιμή μέτρησης από τον μετρητή που εμφανίζεται στην είσοδο του σε μία από τις 8 εξόδους του η οποίες και θα συμβολίζουν την παρούσα κατάσταση.

[Γράψτε εδώ το πρόγραμμά σας:](#)

**Πρόγραμμα 2:** Ο αποκωδικοποιητής καταστάσεων.

--FOTIS STAMATAKIS--

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

*entity state\_decoder is*

```
port(
    Din : in std_logic_vector(2 downto 0);
    Dout : out std_logic_vector(7 downto 0)
);
end state_decoder;
```

*architecture rtl of state\_decoder is*

*begin*

```
process(Din)
begin
    Dout <= (others => '0');
    case Din is
        when "000" => Dout(0) <= '1'; -- T0
        when "001" => Dout(1) <= '1'; -- T1
        when "010" => Dout(2) <= '1'; -- T2
        when "011" => Dout(3) <= '1'; -- T3
        when "100" => Dout(4) <= '1'; -- T4
        when "101" => Dout(5) <= '1'; -- T5
        when "110" => Dout(6) <= '1'; -- T6
        when "111" => Dout(7) <= '1'; -- T7
        when others => null;
    end case;
```

```
end process;  
end rtl;
```

## **Απαριθμητής**

Γράψτε τον κώδικα για έναν μετρητή με εύρος 3-bits με σήματα εισόδου/ελέγχου inc για την αύξηση κατά ένα και rst για εκκαθάριση και σήμα εξόδου count .

[Γράψτε εδώ το πρόγραμμά σας:](#)

**Πρόγραμμα 3:** Ο απαριθμητής των 3-bits.

```
--FOTIS STAMATAKIS--
```

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity counter3 is  
    port(  
        clock : in std_logic;  
        rst  : in std_logic;  
        inc  : in std_logic;  
        count : out std_logic_vector(2 downto 0)  
    );  
end counter3;
```

```

architecture rtl of counter3 is
    signal cnt : std_logic_vector(2 downto 0);
begin
    process(clock)
    begin
        if rising_edge(clock) then
            if rst = '1' then
                cnt <= "000";
            elsif inc = '1' then
                cnt <= cnt + 1;
            end if;
        end if;
    end process;

    count <= cnt;
end rtl;

```

## **Μονάδα Ελέγχου.**

Έχοντας ολοκληρώσει τη συγγραφή του κώδικα για τα επιμέρους στοιχεία που συνθέτουν την μονάδα ελέγχου και αφού όλα συγκεντρωθούν σε μία βιβλιοθήκη, μπορεί πλέον να γραφεί το συνολικό πρόγραμμα περιγραφής της μονάδας ελέγχου. Σημειώνεται εδώ ότι δεδομένου ότι το κύκλωμα παραγωγής των σημάτων ελέγχου τόσο της ΚΜΕ όσο και του μετρητή καταστάσεων (σχήμα 1) είναι εξαιρετικά απλό δεν είναι απαραίτητη η συγγραφή ξεχωριστού στοιχείου για αυτό.

Γράψτε τον κώδικα για τη βιβλιοθήκη (package), με το όνομα `hardwiredlib`, η οποία θα περιέχει τα επιμέρους στοιχεία που συνθέτουν την μονάδα ελέγχου.

[Γράψτε εδώ το πρόγραμμά σας:](#)

**Πρόγραμμα 4:** βιβλιοθήκη στοιχείων για την μονάδα ελέγχου.

--FOTIS STAMATAKIS--



*library ieee;*

*use ieee.std\_logic\_1164.all;*

*package hardwiredlib is*

*constant INOP\_IDX : integer := 0;*

*constant ILDAC\_IDX : integer := 1;*

*constant ISTAC\_IDX : integer := 2;*

*constant IMVAC\_IDX : integer := 3;*

*constant IMOVR\_IDX : integer := 4;*

*constant IJUMP\_IDX : integer := 5;*

*constant IJMPZ\_IDX : integer := 6;*

*constant IJPNZ\_IDX : integer := 7;*

*constant IADD\_IDX : integer := 8;*

*constant ISUB\_IDX : integer := 9;*

*constant IINAC\_IDX : integer := 10;*

*constant ICLAC\_IDX : integer := 11;*

*constant IAND\_IDX : integer := 12;*

*constant IOR\_IDX : integer := 13;*

*constant IXOR\_IDX : integer := 14;*

*constant INOT\_IDX : integer := 15;*

*constant AR\_LOAD : integer := 0;*

*constant AR\_INC : integer := 1;*

*constant PC\_LOAD : integer := 2;*

*constant PC\_INC : integer := 3;*

*constant DR\_LOAD : integer := 4;*

*constant TR\_LOAD : integer := 5;*

```

constant IR_LOAD : integer := 6;
constant R_LOAD : integer := 7;
constant AC_LOAD : integer := 8;
constant Z_LOAD : integer := 9;
constant READ_OP : integer := 10;
constant WRITE_OP : integer := 11;
constant MEM_BUS : integer := 12;
constant BUS_MEM : integer := 13;
constant PC_BUS : integer := 14;
constant DR_BUS : integer := 15;
constant TR_BUS : integer := 16;
constant R_BUS : integer := 17;
constant AC_BUS : integer := 18;
constant AND_OP : integer := 19;
constant OR_OP : integer := 20;
constant XOR_OP : integer := 21;
constant NOT_OP : integer := 22;
constant AC_INC : integer := 23;
constant AC_ZERO : integer := 24;
constant PLUS_OP : integer := 25;
constant MINUS_OP : integer := 26;

end package hardwiredlib;

package body hardwiredlib is
end package body hardwiredlib;

```

Με βάση το σκελετό που ακολουθεί (πρόγραμμα 5) γράψτε τον κώδικα περιγραφής για της μονάδας ελέγχου, δηλαδή της μηχανής πεπερασμένων καταστάσεων, έτσι όπως διαμορφώνεται από τα επιμέρους στοιχεία και το σχήμα 1. Τα σήματα που θα δέχεται σαν είσοδο το κύκλωμα, εκτός των σημάτων clock και reset, θα είναι τα τέσσερα (4) λιγότερο σημαντικά bit του καταχωρητή εντολών

(ir) και η τιμή του καταχωρητή σημαίας (z). Σαν έξοδοι λαμβάνεται το σήμα mOPs που αντιστοιχεί στην κάθε μικροεντολή εύρους 3627-bits.

[Γράψτε εδώ το πρόγραμμά σας:](#)

**Πρόγραμμα 5:** Μονάδα Ελέγχου.

--FOTIS STAMATAKIS--

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use work.hardwiredlib.all;

entity hardwired is

port(

ir : in std\_logic\_vector(3 downto 0);

clock : in std\_logic;

reset : in std\_logic;

Z : in std\_logic;

mOPs : out std\_logic\_vector(26 downto 0)

);

end hardwired;

architecture arc of hardwired is

component instruction\_decoder

port(

Din : in std\_logic\_vector(3 downto 0);

Dout : out std\_logic\_vector(15 downto 0)

);

end component;

component state\_decoder

port(

Din : in std\_logic\_vector(2 downto 0);

Dout : out std\_logic\_vector(7 downto 0)

);

end component;

component counter3

port(

clock : in std\_logic;

rst : in std\_logic;

inc : in std\_logic;

count : out std\_logic\_vector(2 downto 0)

);

end component;

signal IR\_reg : std\_logic\_vector(3 downto 0);

signal instr\_out : std\_logic\_vector(15 downto 0);

signal state\_out : std\_logic\_vector(7 downto 0);

signal count : std\_logic\_vector(2 downto 0);

signal inc, clr : std\_logic;

signal mOPs\_int : std\_logic\_vector(26 downto 0);

signal NOP, LDAC, STAC, MVAC, MOVR, JUMP, JMPZ, JPNZ, ADD, SUB, INAC, CLAC, IAND, IOR, IXOR,  
INOT : std\_logic;

signal FETCH1, FETCH2, FETCH3, NOP1 : std\_logic;

signal LDAC1, LDAC2, LDAC3, LDAC4, LDAC5 : std\_logic;

```
signal STAC1, STAC2, STAC3, STAC4, STAC5 : std_logic;
signal MVAC1, MOVR1 : std_logic;
signal JUMP1, JUMP2, JUMP3 : std_logic;
signal JMPZY1, JMPZY2, JMPZY3, JMPZN1, JMPZN2 : std_logic;
signal JPNZY1, JPNZY2, JPNZY3, JPNZN1, JPNZN2 : std_logic;
signal ADD1, SUB1, INAC1, CLAC1, AND1, OR1, XOR1, NOT1 : std_logic;
signal any_last_state : std_logic;
```

```
alias t0 is state_out(0);
alias t1 is state_out(1);
alias t2 is state_out(2);
alias t3 is state_out(3);
alias t4 is state_out(4);
alias t5 is state_out(5);
alias t6 is state_out(6);
alias t7 is state_out(7);
```

```
begin
```

```
    process(clock)
```

```
    begin
```

```
        if rising_edge(clock) then
```

```
            if mOPs_int(IR_LOAD) = '1' then
```

```
                IR_reg <= ir;
```

```
            end if;
```

```
        end if;
```

```
    end process;
```

*U\_Counter : counter3*

*port map ( clock => clock, rst => clr, inc => inc, count => count );*

*U\_StateDecoder : state\_decoder*

*port map ( Din => count, Dout => state\_out );*

*U\_InstructionDecoder : instruction\_decoder*

*port map ( Din => IR\_reg, Dout => instr\_out );*

*NOP <= instr\_out(0); LDAC <= instr\_out(1); STAC <= instr\_out(2); MVAC <= instr\_out(3);*  
*MOVR <= instr\_out(4); JUMP <= instr\_out(5); JMPZ <= instr\_out(6); JPNZ <= instr\_out(7);*  
*ADD <= instr\_out(8); SUB <= instr\_out(9); INAC <= instr\_out(10); CLAC <= instr\_out(11);*  
*IAND <= instr\_out(12); IOR <= instr\_out(13); IXOR <= instr\_out(14); INOT <= instr\_out(15);*  
*FETCH1 <= t0; FETCH2 <= t1; FETCH3 <= t2;*  
*NOP1 <= NOP and t3;*  
*LDAC1 <= LDAC and t3; LDAC2 <= LDAC and t4; LDAC3 <= LDAC and t5; LDAC4 <= LDAC and t6;*  
*LDAC5 <= LDAC and t7;*  
*STAC1 <= STAC and t3; STAC2 <= STAC and t4; STAC3 <= STAC and t5; STAC4 <= STAC and t6; STAC5*  
*<= STAC and t7;*  
*MVAC1 <= MVAC and t3; MOVR1 <= MOVR and t3;*  
*JUMP1 <= JUMP and t3; JUMP2 <= JUMP and t4; JUMP3 <= JUMP and t5;*  
*JMPZY1 <= JMPZ and Z and t3; JMPZY2 <= JMPZ and Z and t4; JMPZY3 <= JMPZ and Z and t5;*  
*JMPZN1 <= JMPZ and (not Z) and t3; JMPZN2 <= JMPZ and (not Z) and t4;*  
*JPNZY1 <= JPNZ and (not Z) and t3; JPNZY2 <= JPNZ and (not Z) and t4; JPNZY3 <= JPNZ and (not Z)*  
*and t5;*  
*JPNZN1 <= JPNZ and Z and t3; JPNZN2 <= JPNZ and Z and t4;*  
*ADD1 <= ADD and t3; SUB1 <= SUB and t3; INAC1 <= INAC and t3; CLAC1 <= CLAC and t3;*  
*AND1 <= IAND and t3; OR1 <= IOR and t3; XOR1 <= IXOR and t3; NOT1 <= INOT and t3;*

*any\_last\_state* <= NOP1 or LDAC5 or STAC5 or MVAC1 or MOVR1 or JUMP3 or JMPZN2 or JMPZY3 or JPNZN2 or JPNZY3 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1 or OR1 or XOR1 or NOT1;

*clr* <= reset or *any\_last\_state*;

*inc* <= not *any\_last\_state*;

*process*(FETCH1, FETCH2, FETCH3, LDAC1, LDAC2, LDAC3, LDAC4, LDAC5,

STAC1, STAC2, STAC3, STAC4, STAC5, MVAC1, MOVR1, JUMP1, JUMP2, JUMP3,

JMPZY1, JMPZY2, JMPZY3, JMPZN1, JMPZN2, JPNZY1, JPNZY2, JPNZY3, JPNZN1, JPNZN2,

ADD1, SUB1, INAC1, CLAC1, AND1, OR1, XOR1, NOT1)

*begin*

*mOPs\_int* <= (others => '0');

*mOPs\_int*(AR\_LOAD) <= FETCH1 or FETCH3 or LDAC3 or STAC3;

*mOPs\_int*(AR\_INC) <= LDAC1 or STAC1 or JMPZY1 or JPNZY1;

*mOPs\_int*(PC\_LOAD) <= JUMP3 or JMPZY3 or JPNZY3;

*mOPs\_int*(PC\_INC) <= FETCH2 or LDAC1 or LDAC2 or STAC1 or STAC2 or JMPZN1 or JMPZN2 or JPNZN1 or JPNZN2;

*mOPs\_int*(DR\_LOAD) <= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or STAC4 or JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;

*mOPs\_int*(TR\_LOAD) <= LDAC2 or STAC2 or JUMP2 or JMPZY2 or JPNZY2;

*mOPs\_int*(IR\_LOAD) <= FETCH3;

*mOPs\_int*(R\_LOAD) <= MVAC1;

*mOPs\_int*(AC\_LOAD) <= LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1 or OR1 or XOR1 or NOT1;

*mOPs\_int*(Z\_LOAD) <= *mOPs\_int*(AC\_LOAD);

*mOPs\_int*(READ\_OP) <= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;

*mOPs\_int*(WRITE\_OP) <= STAC5;

*mOPs\_int*(MEM\_BUS) <= *mOPs\_int*(READ\_OP);

*mOPs\_int*(BUS\_MEM) <= STAC5;

*mOPs\_int*(PC\_BUS) <= FETCH1 or FETCH3;

*mOPs\_int*(DR\_BUS) <= LDAC2 or LDAC3 or LDAC5 or STAC2 or STAC3 or STAC5 or JUMP2 or JUMP3 or JMPZY2 or JMPZY3 or JPNZY2 or JPNZY3;

```

mOPs_int(TR_BUS) <= LDAC3 or STAC3 or JUMP3 or JMPZY3 or JPNZY3;
mOPs_int(R_BUS) <= MOVR1 or ADD1 or SUB1 or AND1 or OR1 or XOR1;
mOPs_int(AC_BUS) <= STAC4 or MVAC1;
mOPs_int(AND_OP) <= AND1;
mOPs_int(OR_OP) <= OR1;
mOPs_int(XOR_OP) <= XOR1;
mOPs_int(NOT_OP) <= NOT1;
mOPs_int(AC_INC) <= INAC1;
mOPs_int(AC_ZERO) <= CLAC1;
mOPs_int(PLUS_OP) <= ADD1;
mOPs_int(MINUS_OP) <= SUB1;
end process;

mOPs <= mOPs_int;

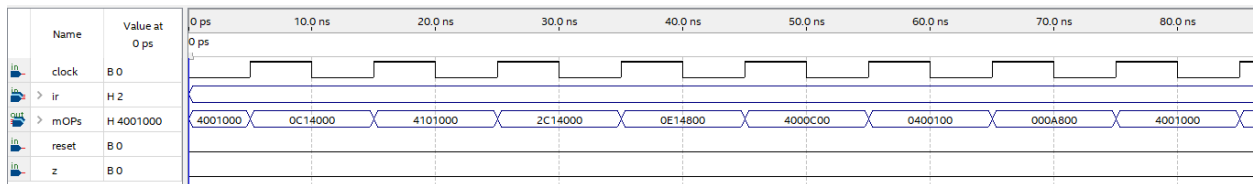
```

*end arc;*

## Εξομοίωση της Μονάδας Ελέγχου.

Το επόμενο στάδιο περιλαμβάνει την εξομοίωση της μονάδας ελέγχου με τον Waveform Editor με σκοπό τον έλεγχο της λειτουργίας της. Με οδηγό τις προηγούμενες ασκήσεις, δημιουργήστε ένα καινούργιο project και εξομοιώστε τη λειτουργία της μονάδας ελέγχου με τη βοήθεια του Waveform Editor για έξι (6) εντολές της KME, της επιλογής σας.

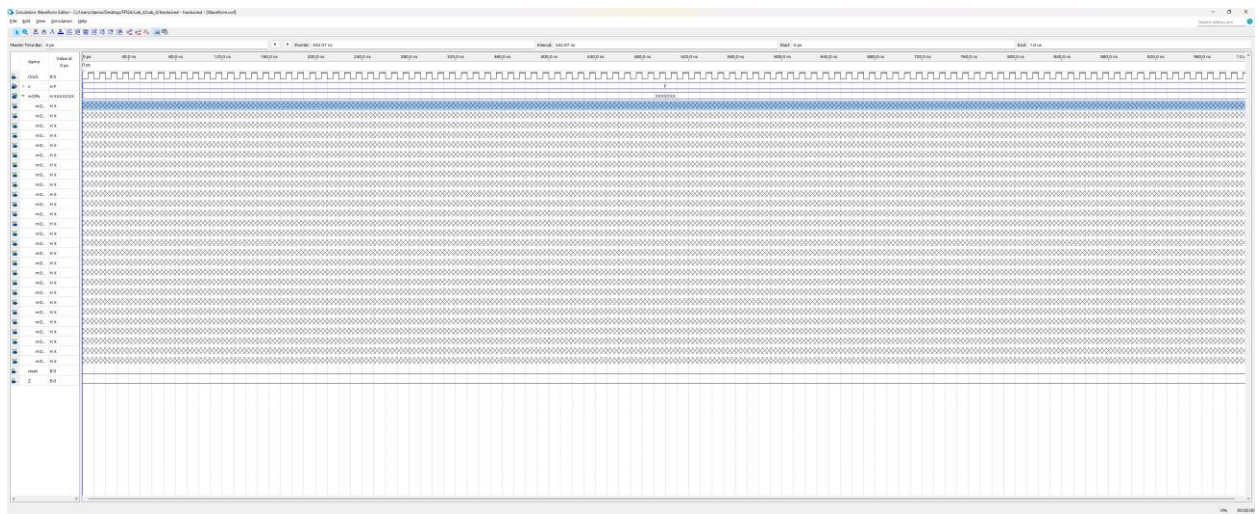
Σαν παράδειγμα ακολουθούν οι κυματομορφές εξομοίωσης για την εντολή STAC (ir=0x2).



**Εικόνα 1:** Κυματομορφές εξομοίωσης εντολής STAC.



**Τοποθετήστε εδώ τις κυματομορφές σας:**



**Εικόνα 2:** Κυματομορφές εξομοίωσης της μονάδας ελέγχου