HARDWIRED
ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

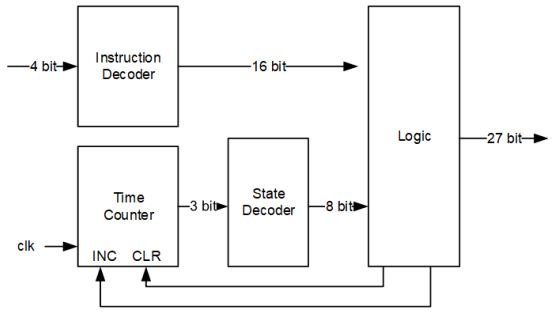
4

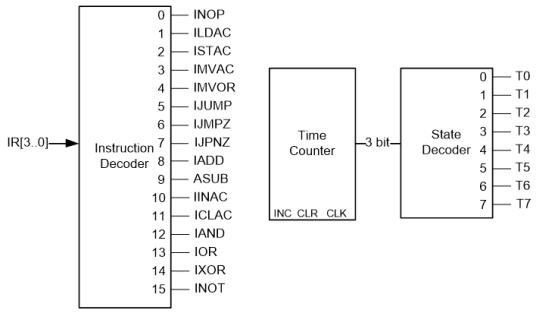
ΓΕΩΡΓΙΟΣ ΜΑΝΤΟΝΙΤΣΑΣ
21183

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 χρησιμοποιούμαι 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 χρησιμοποιούμαι 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 ^ T3
LDAC2 = ILDAC ^ T4

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

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

JUMP3	IJUMP ^ T5		

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

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

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 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 συνδυαστική 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:

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


entity instr_dec is
 port(
  din  : in  std_logic_vector(3 downto 0);
  dout : out std_logic_vector(15 downto 0)
 );
end entity;


architecture rtl of instr_dec is
begin
 process(din)
 begin
  dout <= (others => '0');
  case din is
   when "0000" => dout(0)  <= '1';  --Εντολή INOP
   when "0001" => dout(1)  <= '1';  --Εντολή ILDAC
   when "0010" => dout(2)  <= '1';  --Εντολή ISTAC
   when "0011" => dout(3)  <= '1';  --Εντολή IMVAC
   when "0100" => dout(4)  <= '1';  --Εντολή IMOVR
   when "0101" => dout(5)  <= '1';  --Εντολή IJUMP
   when "0110" => dout(6)  <= '1';  --Εντολή IJMPZ
   when "0111" => dout(7)  <= '1';  --Εντολή IJPNZ
   when "1000" => dout(8)  <= '1';  --Εντολή IADD
```

```
    when "1001" => dout(9)  <= '1';  --Εντολή SUB
    when "1010" => dout(10) <= '1';  --Εντολή IINAC
    when "1011" => dout(11) <= '1';  --Εντολή CLAC
    when "1100" => dout(12) <= '1';  --Εντολή IAND
    when "1101" => dout(13) <= '1';  --Εντολή IOR
    when "1110" => dout(14) <= '1';  --Εντολή IXOR
    when others => dout(15) <= '1';  --Εντολή INOT
   end case;
 end process;
end architecture;
```

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:

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


entity state_dec is
 port(
  din  : in  std_logic_vector(2 downto 0);
  dout : out std_logic_vector(7 downto 0)
 );
end entity;


architecture rtl of state_dec is
begin
 process(din)
 begin
```

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

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:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter3_bit is
 port(
  clock : in  std_logic;
  rst   : in  std_logic;
  inc   : in  std_logic;
  count : out std_logic_vector(2 downto 0)
 );
```

```
end entity;


architecture rtl of counter3_bit is
 signal c : std_logic_vector(2 downto 0);
begin
 process(clock)
 begin
  if rising_edge(clock) then
   if rst = '1' then
    c <= (others => '0');
   elsif inc = '1' then
    c <= c + "001";
   end if;
  end if;
 end process;


 count <= c;
end architecture;
```

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:

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


package hardwiredlib is
```

```
 component instr_dec
  port(
    din  : in  std_logic_vector(3 downto 0);
    dout : out std_logic_vector(15 downto 0)
  );
 end component;

 component state_dec
  port(
    din  : in  std_logic_vector(2 downto 0);
    dout : out std_logic_vector(7 downto 0)
  );
 end component;

 component counter3_bit
  port(
    clock : in  std_logic;
    rst   : in  std_logic;
    inc   : in  std_logic;
    count : out std_logic_vector(2 downto 0)
  );
 end component;

end package;

package body hardwiredlib is

end package body;
```

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:

Λογω το λαθος στοιχειων που δινονται (τουλαχιστον αμα εχω καταλαβει σωστα την ασκηση ειναι λαθος γιατι δεν γινεται να το βγαλω σωστα με τα στοιχεια που δινεται) καθως το FETCH3=T3 που δινεται στον πινακα παραπανω ειναι λαθος καθως το T3 χρησιμοποιηται κανονικα ως αρχικη κατασταση εντολης. Οποτε χρησιμοποιω FETCH3=T2

```
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, reset : in  std_logic;
    z :in  std_logic;
    mOPs : out std_logic_vector(26 downto 0)
  );
end hardwired;

architecture arc of hardwired is
  signal I : std_logic_vector(15 downto 0);
  signal cnt : std_logic_vector(2 downto 0);
  signal T : std_logic_vector(7 downto 0);
  signal T0, T1, T2, T3, T4, T5, T6, T7 : std_logic;
  signal FETCH1, FETCH2, FETCH3 : std_logic;
  signal 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 : std_logic;
  signal JMPZN1, JMPZN2 : std_logic;
  signal JPNZY1, JPNZY2, JPNZY3 : std_logic;
  signal JPNZN1, JPNZN2: std_logic;
  signal ADD1, SUB1, INAC1, CLAC1 : std_logic;
  signal AND1, OR1, XOR1, NOT1 : std_logic;
  signal inc_s : std_logic;
  signal clr_s : std_logic;
  signal z_n : std_logic;
```

```
  signal ARLOAD, ARINC, PCLOAD, PCINC, DRLOAD, TRLOAD, IRLOAD : std_logic;
  signal RLOAD, ACLOAD, ZLOAD, READ, WRITE, MEMBUS, BUSMEM: std_logic;
  signal PCBUS, DRBUS, TRBUS, RBUS, ACBUS : std_logic;
  signal ANDOP, OROP, XOROP, NOTOP, ACINC, ACZERO, PLUS, MINUS : std_logic;

  constant B_ARLOAD : integer := 26;
  constant B_ARINC : integer := 25;
  constant B_PCLOAD: integer := 24;
  constant B_PCINC: integer := 23;
  constant B_DRLOAD : integer := 22;
  constant B_TRLOAD : integer := 21;
  constant B_IRLOAD : integer := 20;
  constant B_RLOAD : integer := 19;
  constant B_ACLOAD : integer := 18;
  constant B_ZLOAD : integer := 17;
  constant B_READ : integer := 16;
  constant B_WRITE : integer := 15;
  constant B_MEMBUS : integer := 14;
  constant B_BUSMEM : integer := 13;
  constant B_PCBUS : integer := 12;
  constant B_DRBUS : integer := 11;
  constant B_TRBUS : integer := 10;
  constant B_RBUS  : integer := 9;
  constant B_ACBUS : integer := 8;
  constant B_ANDOP : integer := 7;
  constant B_OROP : integer := 6;
  constant B_XOROP : integer := 5;
  constant B_NOTOP : integer := 4;
  constant B_ACINC : integer := 3;
  constant B_ACZERO : integer := 2;
  constant B_PLUS : integer := 1;
  constant B_MINUS : integer := 0;

begin

  z_n <= not z;

  U_IDEC : instr_dec
    port map(
      din  => ir,
      dout => I
    );

  U_CNT : counter3_bit
    port map(
      clock => clock,
```

```
    rst   => (reset or clr_s),
    inc   => inc_s,
    count => cnt
  );

U_SDEC : state_dec
  port map(
    din  => cnt,
    dout => T
  );

T0 <= T(0);
T1 <= T(1);
T2 <= T(2);
T3 <= T(3);
T4 <= T(4);
T5 <= T(5);
T6 <= T(6);
T7 <= T(7);

FETCH1 <= T0;
FETCH2 <= T1;
FETCH3 <= T2;
NOP1  <= I(0) and T3;
LDAC1 <= I(1) and T3;
LDAC2 <= I(1) and T4;
LDAC3 <= I(1) and T5;
LDAC4 <= I(1) and T6;
LDAC5 <= I(1) and T7;
STAC1 <= I(2) and T3;
STAC2 <= I(2) and T4;
STAC3 <= I(2) and T5;
STAC4 <= I(2) and T6;
STAC5 <= I(2) and T7;
MVAC1 <= I(3) and T3;
MOVR1 <= I(4) and T3;
JUMP1 <= I(5) and T3;
JUMP2 <= I(5) and T4;
JUMP3 <= I(5) and T5;
JMPZY1 <= I(6) and z  and T3;
JMPZY2 <= I(6) and z  and T4;
JMPZY3 <= I(6) and z  and T5;
JMPZN1 <= I(6) and z_n and T3;
JMPZN2 <= I(6) and z_n and T4;
JPNZY1 <= I(7) and z_n and T3;
JPNZY2 <= I(7) and z_n and T4;
```

```
  JPNZY3 <= I(7) and z_n and T5;
  JPNZN1 <= I(7) and z  and T3;
  JPNZN2 <= I(7) and z  and T4;
  ADD1  <= I(8) and T3;
  SUB1  <= I(9) and T3;
  INAC1 <= I(10) and T3;
  CLAC1 <= I(11) and T3;
  AND1  <= I(12) and T3;
  OR1   <= I(13) and T3;
  XOR1  <= I(14) and T3;
  NOT1  <= I(15) and T3;


  clr_s <= NOP1 or LDAC5 or STAC5 or MVAC1 or MOVR1 or  ADD1 or SUB1 or INAC1 or
CLAC1 or AND1 or OR1 or XOR1 or NOT1 or  JUMP3 or JMPZY3 or JMPZN2 or JPNZY3 or
JPNZN2;
  inc_s <= '1' when (reset = '0' and clr_s = '0') else '0';
  ARLOAD <= FETCH1 or FETCH3 or LDAC3 or STAC3;
  ARINC  <= LDAC1 or STAC1 or JMPZY1 or JPNZY1;
  PCLOAD <= JUMP3 or JMPZY3 or JPNZY3;
  PCINC  <= FETCH2 or LDAC1 or LDAC2 or STAC1 or STAC2 or JMPZN1 or JMPZN2 or
JPNZN1 or JPNZN2;
  DRLOAD <= 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;
  TRLOAD <= LDAC2 or STAC2 or JUMP2 or JMPZY2 or JPNZY2;
  IRLOAD <= FETCH3;
  RLOAD  <= MVAC1;
  ACLOAD <= LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1  or OR1
or XOR1 or NOT1;
  ZLOAD  <= LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1  or OR1
or XOR1 or NOT1;
  READ   <= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1  or STAC2  or JUMP1  or
JUMP2  or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;
  WRITE  <= STAC5;
  MEMBUS <= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1  or STAC2  or JUMP1  or
JUMP2  or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;
  BUSMEM <= STAC5;
  PCBUS  <= FETCH1 or FETCH3;
  DRBUS  <= LDAC2 or LDAC3 or LDAC5 or STAC2 or STAC3 or STAC5 or JUMP2 or
JUMP3 or JMPZY2 or JMPZY3 or JPNZY2 or JPNZY3;
  TRBUS  <= LDAC3 or STAC3 or JUMP3 or JMPZY3 or JPNZY3;
  RBUS   <= MOVR1 or ADD1 or SUB1 or AND1 or OR1 or XOR1;
  ACBUS  <= STAC4 or MVAC1;
  ANDOP  <= AND1;
  OROP   <= OR1;
  XOROP  <= XOR1;
```

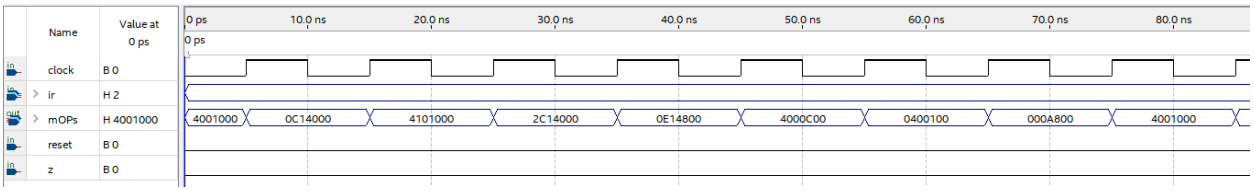
```
  NOTOP  <= NOT1;
  ACINC  <= INAC1;
  ACZERO <= CLAC1;
  PLUS   <= ADD1;
  MINUS  <= SUB1;


  process(all)
    variable v : std_logic_vector(26 downto 0);
  begin
    v := (others => '0');

    v(B_ARLOAD) := ARLOAD;
    v(B_ARINC) := ARINC;
    v(B_PCLOAD) := PCLOAD;
    v(B_PCINC) := PCINC;
    v(B_DRLOAD) := DRLOAD;
    v(B_TRLOAD) := TRLOAD;
    v(B_IRLOAD):= IRLOAD;
    v(B_RLOAD) := RLOAD;
    v(B_ACLOAD) := ACLOAD;
    v(B_ZLOAD) := ZLOAD;
    v(B_READ)  := READ;
    v(B_WRITE) := WRITE;
    v(B_MEMBUS) := MEMBUS;
    v(B_BUSMEM) := BUSMEM;
    v(B_PCBUS) := PCBUS;
    v(B_DRBUS) := DRBUS;
    v(B_TRBUS) := TRBUS;
    v(B_RBUS)  := RBUS;
    v(B_ACBUS) := ACBUS;
    v(B_ANDOP) := ANDOP;
    v(B_OROP)  := OROP;
    v(B_XOROP) := XOROP;
    v(B_NOTOP) := NOTOP;
    v(B_ACINC) := ACINC;
    v(B_ACZERO) := ACZERO;
    v(B_PLUS)  := PLUS;
    v(B_MINUS) := MINUS;

    mOPs <= v;
  end process;

end arc;
```

Πρόγραμμα 1: *Μονάδα Ελέγχου.*

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 ΚΜΕ , 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*