

ΣΧΕΤΙΚΑ ΑΠΛΗ ΚΕΝΤΡΙΚΗ ΜΟΝΑΔΑ ΕΠΕΞΕΡΓΑΣΙΑΣ (ΚΜΕ)

5

Όνοματεπώνυμο : Παναγιώτης Αναγνωστόπουλος

ΑΜ : 9093202100034

Ημ/νία : 30/12/2025

Σκοπός

Με αφορμή την σχεδίαση και την εξομοίωση με διάφορους τρόπους, απλών ψηφιακών κυκλωμάτων θα κατακτηθεί το αντικείμενο της άσκησης αυτής που είναι η τελική σύνθεση της σχετικά απλής ΚΜΕ.

Η εσωτερική δομή της ΚΜΕ περιλαμβάνει τρία βασικά τμήματα:

- Το τμήμα των καταχωρητών (Register Section) στο οποίο, όπως υποδηλώνει και η ονομασία του, περιλαμβάνει μία σειρά από καταχωρητές και ένα μηχανισμό επικοινωνίας μεταξύ τους (εσωτερικός δίαυλος δεδομένων).
- Το τμήμα της αριθμητικής και λογικής μονάδας (Arithmetic Logic Unit - ALU), το οποίο εκτελεί αριθμητικές πράξεις όπως πρόσθεση και λογικές διεργασίες όπως AND, OR κτλ. Λαμβάνει τελεστές από το τμήμα καταχωρητών της ΚΜΕ, εκτελεί τις ανάλογες λογικές ή αριθμητικές πράξεις και αποθηκεύει τα αποτελέσματα στο τμήμα καταχωρητών.
- Το τμήμα της μονάδας ελέγχου (Control Unit) το οποίο είναι υπεύθυνο για τον έλεγχο κάθε λειτουργίας του επεξεργαστή. Η μονάδα ελέγχου λαμβάνει κάποιες τιμές δεδομένων από το τμήμα των καταχωρητών, τις οποίες χρησιμοποιεί για να δημιουργήσει τα σήματα ελέγχου. Ο ρόλος της μονάδας ελέγχου είναι να τοποθετήσει αυτά τα σήματα ελέγχου στη σωστή σειρά, ώστε η CPU αλλά και το υπόλοιπο του υπολογιστή να εκτελέσουν τις διεργασίες που απαιτούνται για την σωστή επεξεργασία των εντολών και διακοπών.

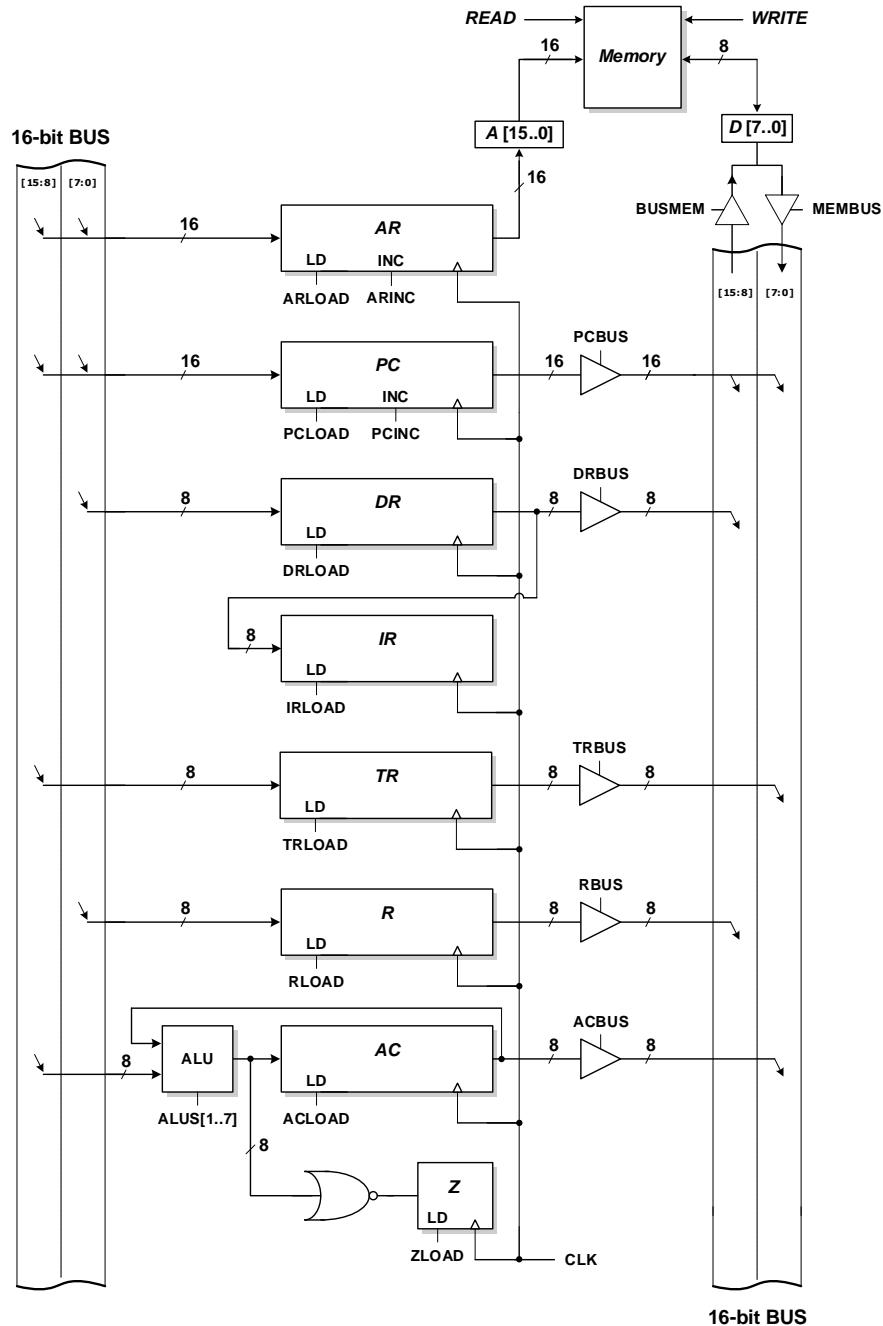
Το σύνολο των καταχωρητών αποτελεί μέρος της ISA μιας ΚΜΕ, αφού βάσει αυτών καθορίζεται ο τρόπος εκτέλεσης κάθε εντολής. Για την περίπτωση της σχετικά απλής ΚΜΕ, οι διαθέσιμοι για τη σχεδίαση και υλοποίησή της καταχωρητές είναι οι εξής :

- Ένας καταχωρητής διεύθυνσης των 16-bits (Address Register - AR)
- Ένας απαριθμητής προγράμματος των 16-bits (Program Counter - PC)
- Ένας καταχωρητής δεδομένων των 8-bits (Data Register - DR)
- Ένας καταχωρητής εντολών των 8-bits (Instruction Register - IR)

- Ένας συσσωρευτής των 8-bits (Accumulator - AC)
- Ένας προσωρινός καταχωρητής των 8-bits (Temporary Register - TR)
- Ένας καταχωρητής σημαίας του 1-bit (Flag register - Z)
- Ένας καταχωρητής γενικού σκοπού των 8-bits (Register - R)

Ο καταχωρητής διεύθυνσης (AR) είναι εύρους 16-bits και είναι αυτός που παρέχει τη διεύθυνση μνήμης από την οποία θα γίνει η ανάγνωση μιας εντολής ή ενός δεδομένου. Ομοίως ο απαριθμητής προγράμματος (PC) είναι εύρους 16-bits και είναι αυτός που παρέχει την διεύθυνση της επόμενης προς εκτέλεση εντολής στον καταχωρητή διεύθυνσης. Ο καταχωρητής δεδομένων (DR) έχει εύρος 8-bits και εκτός από το να δέχεται δεδομένα από τη μνήμη (DR<-M), μπορεί και να γράψει δεδομένα σε αυτήν (M<-DR). Επιπρόσθετα, ο καταχωρητής εντολών (IR) έχει εύρος 8-bits, όπως και ο συσσωρευτής (AC), όπου ο μεν πρώτος χρησιμοποιείται για την αποθήκευση του τμήματος της εντολής που αφορά την επιλογή ενός από τους δεκαέξι (16) διαθέσιμους κύκλους εκτέλεσης (execution cycle), ενώ ο δεύτερος χρησιμοποιείται όπως είναι γνωστό για την καταχώρηση του αποτελέσματος μιας διεργασίας στην ALU.

Το συνολικό data path όπου απεικονίζονται οι απαραίτητες διασυνδέσεις των καταχωρητών, της εξωτερικής μνήμης, των απομονωτών, της ALU και ο αριθμός των bits σε κάθε δίαυλο φαίνονται στο σχήμα 1. Για λόγους απλότητας έχει παραληφθεί το τμήμα της μονάδας ελέγχου η οποία παρέχει όλα τα απαραίτητα σήματα ελέγχου.



Σχήμα 1: Συνολικό data path σχετικά απλίξ MKE.

Κύκλωμα παραγωγής σημάτων ALU

Όπως έχει αναφερθεί ο τμήμα της μονάδας ελέγχου παρέχει όλα τα απαραίτητα σήματα ελέγχου απευθείας στα επιμέρους στοιχεία της KME με μοναδική εξαίρεση την ALU. Για την σωστή λειτουργεία της ALU είναι απαραίτητη παραγωγή των σημάτων ελέγχου alus[0..7] από τα σήματα που παράγει η μονάδα ελέγχου. Το κύκλωμα αυτό δίνεται στο πρόγραμμα 1 που ακολουθεί.

```
library ieee;
```

```

use ieee.std_logic_1164.all ;

entity alus IS
port(rbus,acload,zload,andop
      orop,notop,xorop,aczero
      acinc,plus,minus,drbus
      alus
      : in std_logic;
      : in std_logic;
      : in std_logic;
      : out std_logic_vector(6 downto 0));
end alus ;

architecture arc of alus is
signal control : std_logic_vector(11 downto 0);
begin
control <= rbus & drbus & acload & zload & andop & orop
& notop & xorop & aczero & acinc & plus & minus ;
process(control)
begin
case control is
WHEN "101110000000" => alus <= "1000000" ;-- AND
WHEN "101101000000" => alus <= "1100000" ;-- OR
WHEN "001100100000" => alus <= "1110000" ;-- NOT
WHEN "101100010000" => alus <= "1010000" ;-- XOR
WHEN "001100001000" => alus <= "0000000" ;-- CLAC
WHEN "001100000100" => alus <= "0001001" ;-- INAC
WHEN "101100000010" => alus <= "0000101" ;-- ADD
WHEN "101100000001" => alus <= "0001011" ;-- SUB
WHEN "101100000000" => alus <= "0000100" ;-- MOVR
WHEN "011100000000" => alus <= "0000100" ;-- LDAC5
WHEN others => alus <= "1111111" ;-- NO OPERATION
end case;
end process;
end arc ;

```

Πρόγραμμα 1: Κύκλωμα παραγωγής σημάτων ελέγχου ALU.

Δίαυλος Δεδομένων

Ο δίαυλος δεδομένων, σε συνδυασμό με τη διάταξη των απομονωτών (buffers), παίζει σημαντικό ρόλο σε μια KME, αφού η λειτουργία της εξαρτάται από τη σωστή οργάνωση της διακίνησης των δεδομένων στο εσωτερικό της. Με εξαίρεση τους καταχωρητές IR και Z, όλοι οι υπόλοιποι δέχονται δεδομένα μέσω του διαύλου, ενώ πέντε (5) από αυτούς παρέχουν το περιεχόμενό τους σε αυτόν, μέσω απομονωτών. Παράλληλα, δυνατότητα αποστολής και λήψης δεδομένων μέσω του διαύλου έχει και η μνήμη.

Γράψτε τον κώδικα για τον δίαυλο δεδομένων με τους απομονωτές όπως προκύπτει από το σχήμα 1.

Γράψτε εδώ το πρόγραμμά σας:

Πρόγραμμα 2: Ο δίαυλος δεδομένων.

```

library ieee;
use ieee.std_logic_1164.all;

entity data_bus is
port(
-- data
pc_q : in std_logic_vector(7 downto 0);
dr_q : in std_logic_vector(7 downto 0);
tr_q : in std_logic_vector(7 downto 0);
r_q : in std_logic_vector(7 downto 0);
ac_q : in std_logic_vector(7 downto 0);
mem_q : in std_logic_vector(7 downto 0);

-- buffers (enables)
pcbus : in std_logic;
drbus : in std_logic;
trbus : in std_logic;
rbus : in std_logic;
acbus : in std_logic;
membus : in std_logic;

-- outputs
dataBus_o : out std_logic_vector(7 downto 0);
busmem : out std_logic_vector(7 downto 0)
);
end entity data_bus;

architecture rtl of data_bus is
begin
process(pc_q, dr_q, tr_q, r_q, ac_q, mem_q, pcbus, drbus, trbus, rbus, acbus, membus)
variable vbus : std_logic_vector(7 downto 0);
begin
vbus := (others => '0');

if pcbus = '1' then vbus := pc_q;
elsif drbus = '1' then vbus := dr_q;

```

```

elsif trbus = '1' then vbus := tr_q;
elsif rbus = '1' then vbus := r_q;
elsif acbus = '1' then vbus := ac_q;
elsif membus = '1' then vbus := mem_q;
else
  vbus := (others => '0'); -- idle bus
end if;

dataBus_o <= vbus;
busmem <= vbus;
end process;
end architecture rtl;

```

Η Εξωτερική Μνήμη

Το τμήμα που ολοκληρώνει την υλοποίηση της σχετικά απλής KME σε VHDL, είναι η εξωτερική μνήμη (external memory) ή απλά μνήμη η οποία αποτελεί την πηγή παροχής δεδομένων και εντολών για τη λειτουργία της MKE δηλαδή περιέχει τα προγράμματα εφαρμογών .

Η υλοποίηση της εξωτερικής μνήμης σε VHDL, θα γίνει μέσω του δομικού στοιχείου του Quartus , RAM: 1-PORT με τη βοήθεια του Megafunction Wizard του Quartus.

Με τη βοήθεια οποιουδήποτε Editor συντάξτε και αποθηκεύστε το πρόγραμμα 3 που ακολουθεί με όνομα "extRAM.mif". Το αρχείο αυτό θα χρησιμοποιηθεί για την αρχικοποίηση της μνήμης

μικροκώδικα(μεταβλητή lpm_file) που θα δημιουργήσουμε στο επόμενο βήμα και περιέχει το σύνολο των εντολών του προγράμματος που θα εκτελέσει η KME.

```
WIDTH=8;
DEPTH=256;

ADDRESS_RADIX=DEC;
DATA_RADIX=BIN;

CONTENT BEGIN
    0 : 00000001; -- LDAC
    1 : 00101000; -- 28H
    2 : 00000000; -- 00H
    3 : 00000010; -- STAC
    4 : 00101011; -- 2BH
    5 : 00000000; -- 00H
    6 : 00000011; -- MVAC
    7 : 00001010; -- INAC
    8 : 00001111; -- NOT
    9 : 00001110; -- XOR
   10 : 00001000; -- ADD
   11 : 00001001; -- SUB
   12 : 00000100; -- MOVR
   13 : 00000101; -- JUMP
   14 : 00011000; -- 18H
   [15..23] : 00000000;
   24 : 00000111; -- JPNZ
   25 : 00100000; -- 20H
   26 : 00000000; -- 00H
   27 : 00001111; -- NOT
   28 : 00000111; -- JPNZ
   [29..31] : 00000000;
   32 : 00001011; -- CLAC
   33 : 00000110; -- JMPZ
   34 : 00011000; -- 18H
   [35..39] : 00000000;
   40 : 01111010; -- 7AH
   [41..255] : 00000000;
END;
```

Πρόγραμμα 3: Περιεχόμενα μνήμης

Ακολουθώντας τη διαδικασία που έχει περιγραφεί στην άσκηση 3 δημιουργήστε μια μνήμη RAM – 1PORT με 256 θέσεις , εύρους 8-bits η κάθε μία,

Γράψτε εδώ το πρόγραμμά σας:

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

library altera_mf;
```

```

use altera_mf.altera_mf_components.all;

entity extRAM is
port(
    clk : in std_logic;
    addr : in std_logic_vector(7 downto 0); -- 0..255
    we : in std_logic; -- write enable
    din : in std_logic_vector(7 downto 0); -- data to memory
    dout : out std_logic_vector(7 downto 0) -- data from memory
);
end entity;

architecture rtl of extRAM is
begin

u_ram : altsyncram
generic map(
    operation_mode      => "SINGLE_PORT",
    width_a            => 8,
    widthad_a          => 8,
    numwords_a         => 256,
    outdata_reg_a      => "UNREGISTERED",
    init_file          => "extRAM.mif",
    intended_device_family => "Cyclone V",
    read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ"
)
port map(
    clock0  => clk,
    address_a => addr,
    wren_a   => we,
    data_a   => din,
    q_a      => dout
);

end architecture;

```

Πρόγραμμα 4: Η εξωτερική μνήμη.

Συνολική Υλοποίηση KME

Έχοντας ολοκληρώσει την περιγραφή του κώδικα σε VHDL για κάθε ένα επιμέρους τμήμα της MKE, και αφού όλα συγκεντρωθούν σε μία βιβλιοθήκη, μπορεί πλέον να γίνει η διασύνδεσή τους σε ένα κεντρικό πρόγραμμα.

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

Γράψτε εδώ το πρόγραμμά σας:

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

package cpulib is

component alus is
port(
    rbus, zload, zload, andop, orop, notop, xorop, aczero, acinc, plus, inus, drbus: in std_logic;
    alus: out std_logic_vector(6 downto 0);
);
end component;

component data_bus is
port(
    pc_q : in std_logic_vector(7 downto 0);
    dr_q : in std_logic_vector(7 downto 0);
    tr_q : in std_logic_vector(7 downto 0);
    r_q : in std_logic_vector(7 downto 0);
    ac_q : in std_logic_vector(7 downto 0);
    mem_q : in std_logic_vector(7 downto 0);

    pcbus : in std_logic;
    drbus : in std_logic;
    trbus : in std_logic;
    rbus : in std_logic;
```

```

acbus : in std_logic;
membus : in std_logic;

dataBus_o : out std_logic_vector(7 downto 0);
busmem  : out std_logic_vector(7 downto 0)
);

end component;

component extRAM is
port(
clk : in std_logic;
addr : in std_logic_vector(7 downto 0);
we  : in std_logic;
din : in std_logic_vector(7 downto 0);
dout : out std_logic_vector(7 downto 0)
);
end component;

end package cpulib;

package body cpulib is
end package body cpulib;

```

Πρόγραμμα 5: βιβλιοθήκη στοιχείων για την KME.

Με βάση το σκελετό που ακολουθεί (πρόγραμμα 6) γράψτε τον κώδικα περιγραφής για την KME. Σαν εξωτερικά σήματα εκτός των clock και reset, να οριστούν τα δεδομένα των καταχωρητών (ARdata, PCdata, DRdata, ACdata, IRdata, RRdata και TRdata), η τιμή της σημαίας (ZRdata), τα δεδομένα των διαύλων διευθύνσεων και δεδομένων (addressBus, dataBus) καθώς και το τμήμα των μικρολειτουργιών μOPs (mOP) με σκοπό τον έλεγχό τους σε κάθε παλμό.

ΣΗΜΕΙΩΣΗ: Χρησιμοποιήστε όποια μονάδα ελέγχου (*microprogrammed* ή *hardwired*) επιθυμείτε.

Γράψτε εδώ το πρόγραμμά σας:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.cpulib.all;

entity rs_cpu is
port(
    ARdata, PCdata : buffer std_logic_vector(15 downto 0);
    DRdata, ACdata : buffer std_logic_vector(7 downto 0);
    IRdata, TRdata : buffer std_logic_vector(7 downto 0);
    RRdata      : buffer std_logic_vector(7 downto 0);
    ZRdata      : buffer std_logic;
    clock, reset : in std_logic;
    mOP         : buffer std_logic_vector(26 downto 0);
    addressBus  : buffer std_logic_vector(15 downto 0);
    dataBus     : buffer std_logic_vector(7 downto 0)
);
end rs_cpu;
```

architecture arc of rs_cpu is

```
-- εσωτερικά σήματα RAM / bus
signal mem_q   : std_logic_vector(7 downto 0);
signal busmem   : std_logic_vector(7 downto 0);
signal dataBus_o : std_logic_vector(7 downto 0);

-- enables προς data_bus
signal pcbus_s, drbus_s, trbus_s, rbus_s, acbus_s, membus_s : std_logic := '0';

-- extRAM interface
signal ram_addr  : std_logic_vector(7 downto 0);
signal ram_we    : std_logic := '0';
signal ram_din   : std_logic_vector(7 downto 0);
```

```

-- alus

signal alus_s : std_logic_vector(6 downto 0);

signal acload_s, zload_s, andop_s, orop_s, notop_s, xorop_s, aczero_s, acinc_s, plus_s, minus_s : std_logic := '0';

-- fetch

type state_t is (S_RESET, S_FETCH1, S_FETCH2);

signal state : state_t := S_RESET;

begin

addressBus <= ARdata;

dataBus  <= dataBus_o;

ram_addr <= ARdata(7 downto 0);

ram_din <= busmem;

-- External RAM

u_ram : extRAM

port map(
    clk => clock,
    addr => ram_addr,
    we  => ram_we,
    din  => ram_din,
    dout => mem_q
);

-- Data bus mux

u_bus : data_bus

port map(
    pc_q  => PCdata(7 downto 0),
    dr_q  => DRdata,
    tr_q  => TRdata,
    r_q   => RRdata,
    ac_q  => ACdata,
    mem_q => mem_q,
);

```

```

pcbus => pcbus_s,
drbus => drbus_s,
trbus => trbus_s,
rbus => rbus_s,
acbus => acbus_s,
membus => membuss,
dataBus_o => dataBus_o,
busmem => busmem
);

```

```

-- ALU control
u_alus : alus
port map(
    rbus => rbus_s,
    acload => acload_s,
    zload => zload_s,
    andop => andop_s,
    orop => orop_s,
    notop => notop_s,
    xorop => xorop_s,
    aczero => aczero_s,
    acinc => acinc_s,
    plus => plus_s,
    minus => minus_s,
    drbus => drbus_s,
    alus => alus_s
);

```

```

-- Hardwired control
process(clock, reset)
begin
if reset = '1' then
    state <= S_RESET;
    ARdata <= (others => '0');
    PCdata <= (others => '0');

```

```
IRdata <= (others => '0');

DRdata <= (others => '0');

ACdata <= (others => '0');

RRdata <= (others => '0');

TRdata <= (others => '0');

ZRdata <= '0';
```

```
elsif rising_edge(clock) then
```

```
    pcbus_s <= '0';
    drbus_s <= '0';
    trbus_s <= '0';
    rbus_s <= '0';
    acbus_s <= '0';
    membus_s <= '0';
    ram_we <= '0';
```

```
    mOP <= (others => '0');
```

```
case state is
```

```
when S_RESET =>
    state <= S_FETCH1;
```

```
when S_FETCH1 =>
```

```
    ARdata <= PCdata;
```

```
    mOP(0) <= '1';
```

```
    state <= S_FETCH2;
```

```
when S_FETCH2 =>
```

```
    membus_s <= '1';
    IRdata <= mem_q;
```

```

PCdata <= std_logic_vector(unsigned(PCdata) + 1);

if ACdata = x"00" then
    ZRdata <= '1';
else
    ZRdata <= '0';
end if;

mOP(1) <= '1';
mOP(2) <= '1';

state <= S_FETCH1;

end case;
end if;
end process;

end arc;

```

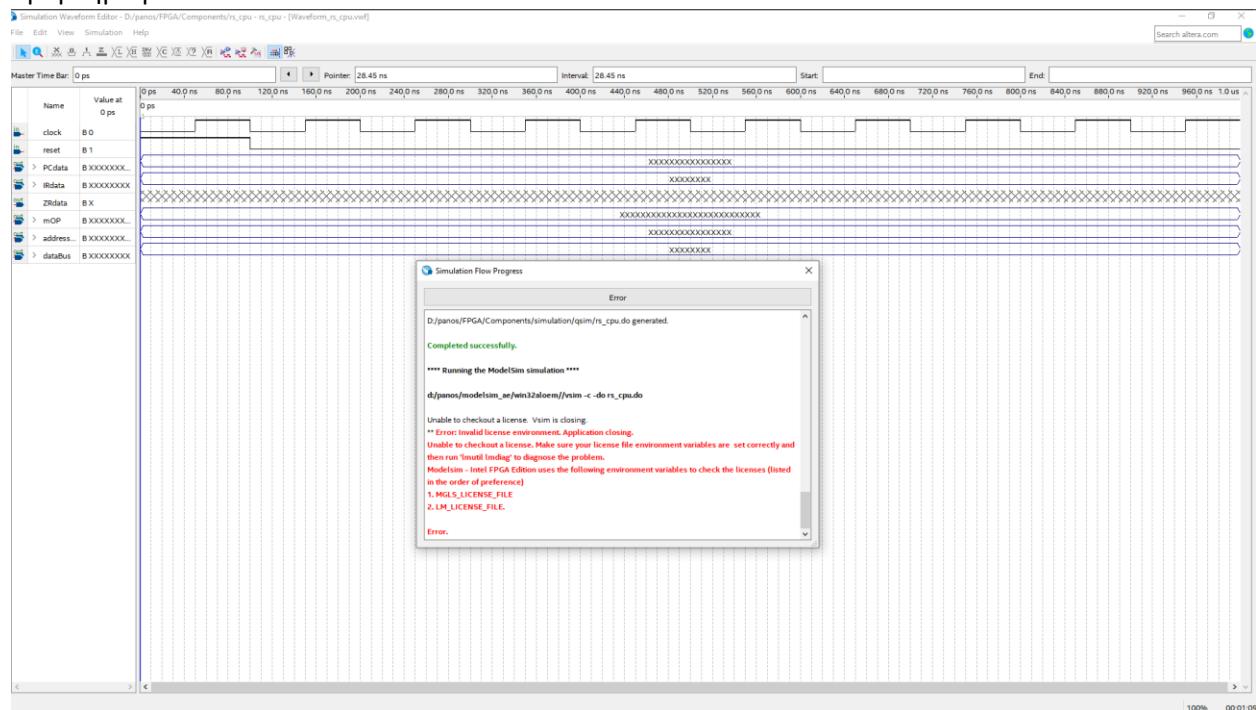
Πρόγραμμα 6: Συνολική περιγραφή της KME.

Εξομοίωση της KME.

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 KME.

Τοποθετήστε εδώ τις κυματομορφές σας:

Δεν μπορούσε να υλοποιηθεί η κυματομορφή, διότι δεν μπορούσε να γίνει εξομοίωση μετά από πρόβλημα με το licence του ModelSim



Εικόνα 1: Κυματομορφές εξομοίωσης της KME

Δώστε την ανάλυση των περιεχομένων της εξωτερικής μνήμης που χρησιμοποιήσατε (πρόγραμμα 3) συμπληρώνοντας τον ακόλουθο πίνακα:

A/A Μνήμης	Περιεχόμενο(Hex)	Περιεχόμενο(Bin)	Εντολή	Λειτουργία
------------	------------------	------------------	--------	------------

0	01	00000001	LDAC	Αμεση φόρτωση τιμής στον AC
1	28	00101000	-	Operand για LDAC
40	7A	01111010	-	Δεδομένο στην μνήμη
3	02	00000010	STAC	Αποθήκευση του AC (περιχόμενο του AC) στην μνήμη
4	2B	00101011	-	Διεύθυνση μνήμης για STAC
6	03	00000011	MVAC	Κίνηση δεδομένων από AC
7	0A	00001010	INAC	AC <- AC+1
8	0F	00001111	NOT	NOT
9	0E	00001110	XOR	XOR με άλλο τελεστή
10	08	00001000	ADD	ADD με άλλο τελεστή
11	09	00001001	SUB	SUB με άλλο τελεστή
12	04	00000100	MOVR	Μεταφορά δεδομένων μεταξύ AC και R
13	05	00000101	JUMP	
14	16	00010110	-	Jump σε διεύθυνση
22	07	00000111	JPNZ	Jump υπό συνθήκη Z=0
23	20	00100000	-	Jump σε διεύθυνση
32	0B	00001011	CLAC	AC = 0
33	06	00000110	JMPZ	Jump υπό συνθήκη Z=1
34	1C	00011100	AND B	AC AND B
25	01	00000001	LDAC	LOAD AC (IM)
28	07	00000111	JPNZ	Jump υπό συνθήκη Z=0

Δώστε τα συμπερασματά σας και τις παρατηρήσεις σας σχετικά με τη λειτουργία της σχετικά απλής CPU. Είναι τα αποτελέσματα της εξομοίωσης τα αναμενόμενα σύμφωνα με τον κώδικα της εξωτερικής μνήμης

Γράξτε εδώ τα σχόλια σας:

Από την ανάλυση του Προγράμματος 3 προκύπτει ότι η CPU εκτελεί σωστά όλες τις βασικές κατηγορίες εντολών: φόρτωση και αποθήκευση δεδομένων στη μνήμη, αριθμητικές και λογικές πράξεις μέσω της ALU, καθώς και έλεγχο ροής με άλματα. Έτσι επιβεβαιώνεται ότι η μονάδα ελέγχου και το datapath συνεργάζονται ορθά.

Παρότι δεν παρουσιάζονται κυματομορφές εξομοίωσης λόγω του προβλήματος που αναφέραμε και πριν, η αναμενόμενη λειτουργία της CPU μπορεί να επαληθευτεί θεωρητικά από την ακολουθία εντολών της εξωτερικής μνήμης. Η ροή εκτέλεσης και τα αποτελέσματα συμφωνούν με τον σχεδιασμό και τις λειτουργίες που υλοποιήθηκαν στον κώδικα της KME.

