ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

ΗΡΥ 312

ΕΡΓΑΣΤΗΡΙΟ 1

Αναφορά Εργαστηριακής Άσκησης

Ομάδα Εργασίας LAB31220188 (**2 ατόμων**) : Οργάνωση Υπολογιστών - HPY312			
α/α	**Α.Μ.**	**Ονοματεπώνυμο**	
1	2011030010	Χριστοδουλου Θεοφιλος	
2	2011030009	ΚΑΡΙΜΠΙΔΗΣ ΔΙΟΝΥΣΗΣ	

Το 1ο εργαστήριο περιείχε 2 μέρη προς υλοποίηση.

Α) Στο πρώτο μέρος κληθήκαμε να δημιουργήσουμε μία Μονάδα Αριθμητικών και Λογικών Πράξεων (**ALU**) , η οποία εκτελεί τις βασικές αριθμητικές πράξεις (**πρόσθεση-αφαίρεση**), καθώς και κάποιες λογικές (**NOT, AND, OR, shift, rotate**), μεταξύ δυο δυαδικών τελεστέων. Ταυτόχρονα, σήματα μας ενημερώνουν για τις ενδιαφέρουσες περιπτώσεις που μπορεί να προκύψουν από την εκτέλεση μιας πράξης, όπως αν υπάρχει μηδενικό αποτέλεσμα(**Zero**), υπερχείληση(overflow- **Ovf**) ή κρατούμενο (carry out- **Cout**).

Β) Στο Β' μέρος το ζητούμενο ήταν να υλοποιήσουμε -αρχικά- ένα σύγχρονο **32-bit register**, με λειτουργία ανάγνωσης και εγγραφής. Σε δεύτερη φάση, συνδέσαμε 32 τέτοιους register, δημιουργώντας ένα Αρχείο Καταχωρητών (**Register File**), που υλοποιεί μια μνήμη **32 θέσεων** των 32 bit, με δυνατότητα **εγγραφής** και **ανάγνωσης** (ακόμα και από 2 θέσεις μνήμης) . Ειδική μέριμνα σε περίπτωση ανάγνωσης καταχωρητή την ίδια στιγμή που γίνεται εγγραφή σε αυτόν.

1) Σχεδίαση της μονάδας αριθμητικών και λογικών πράξεων (ALU)

Για την υλοποίηση της ALU χρησιμοποιήσαμε 1 module που έχει ώς **εισόδους** τα **A**(32bits) και **B**(32bits) και άλλη μια είσοδο **Op**(4bits) που υποδηλώνει τη πράξη θα γίνει ανάμεσα στα A και B. Για να υλοποιήσουμε λοιπόν εύκολα τις πράξεις χρησιμοποιήσαμε επιπλέον τις βιβλιοθήκες:

```
use IEEE.STD_LOGIC_SIGNED.ALL;
use ieee.std_logic_arith.all;
```

και δημιουργήσαμε και 2 signals:

temp : std_logic_vector (31 downto 0) όπου αποθηκεύουμε προσωρινά το αποτέλεσμα της πράξης και στην συνέχεια εκχωρούμε την τιμή του στην **έξοδο Output**(32bits) της ALU

b_minus : std_logic_vector (31 downto 0) όπου το χρησιμοποιούμε για να συγκρατήσουμε την τιμή του -B ώστε να κάνουμε την πράξη της αφαίρεσης.

Γνωρίζουμε ότι για να αφαιρέσουμε από το A το B, ουσιαστικά θα του προσθέσουμε το -B. Αυτός είναι και ο λόγος που χρησιμοποιούμε το signal αυτό ενώ θα μπορούσαμε να γράψουμε κατευθείαν την εντολή (A-B) αντί της (A+b_minus).

Στην συνέχεια εκτελούμε την κατάλληλη πράξη ανάλογα το Op code που δίνεται.

Για **πρόσθεση**: *temp <= (A+B)* μπορούμε να γραψουμε με αυτόν τον τρόπο την εντολή για την πρόσθεση λόγω των βιβλιοθηκών που χρησιμοποιήσαμε

Για **αφαίρεση**: *temp <= A+b_minus*

Λογικό **AND**: *temp <= A and B*

Αντιστροφή του A(**!A**): temp <= not A

Λογικό Ή(**OR**): temp <= A or B

Λογική ολίσθηση δεξιά κατά 1 θέση(**Shift Left** 1bit): temp <= '0' & A(31 downto 1)
κρατάμε τα 31 “κάτω” bits και στο MSB θέτουμε το '0'

Λογική ολίσθηση αριστερά κατά 1 θέση(**Shift Right** 1bit): temp <= A(30 downto 0) & '0'
κρατάμε τα 31 “πανω” bits και στο LSB θέτουμε το '0'

Κυκλικό rotate αριστερά κατα 1 θέση(**Rotate Left** 1bit): temp <= A(30 downto 0) & A(31)
το MSB γίνεται LSB

Κυκλικό rotate δεξιά κατα 1 θέση: (**Rotate Right** 1bit) temp <= A(0) & A(30 downto 0)
το LSB γίνεται MSB

Καριμπίδης Διονύσης **A.M. : 2011030009**
Χριστοδούλου Θεόφιλος **A.M. : 2011030010**
Μάρτιος 2014

Τέλος ελέγχουμε 3 συνθήκες:

Zero: Αν το αποτέλεσμα της πράξης ήταν 0 (δηλαδή $temp = 0$) ενεργοποιούμε το σήμα $zero <= '1'$ αλλιώς $zero <= '0'$.

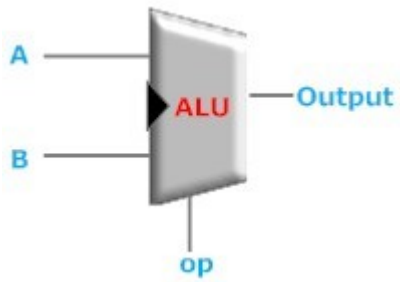
Overflow: Αν είχαμε overflow. Overflow έχουμε όταν το αποτέλεσμα της πρόσθεσης 2 *ομόσημων αριθμών* μας δώσει *ετερόσημο αποτέλεσμα*. Έτσι ελέγχουμε με μια *XOR* το πρόσημο των A και B που θα μας δώσει '0' αν τα πρόσημα είναι ίδια ή '1' αν τα πρόσημα είναι διαφορετικά και με μια άλλη *XOR* το πρόσημο των A και temp(άθροισμα).

Carry _out: Αν έχουμε carry out. Ελέγχουμε και σε αυτήν την περίπτωση με μια *XOR* το πρόσημο των A και B. *Αν αυτό είναι ίδιο το Cout θα είναι ίσο με το πρόσημο του A ή του B, αλλιώς θα είναι το not temp(31)*. Αυτό φαίνεται ξεκάθαρα στον παρακάτω πίνακα:

A(31)	B(31)	temp(31)	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Παρακάτω φαίνεται και ο πίνακας λειτουργιών,όπως αυτές καθορίζονται σύμφωνα με το εκάστοτε OP code :

OP code	Λειτουργία
1111	Πρόσθεση A+B
0111	Αφαίρεση A-B
0011	A **AND** B
0001	**NOT** A
0000	A **OR** B
1000	Shift Right (A)
1100	Shift Left (A)
1110	Rotate Left (A)
0101	Rotate Right (A)



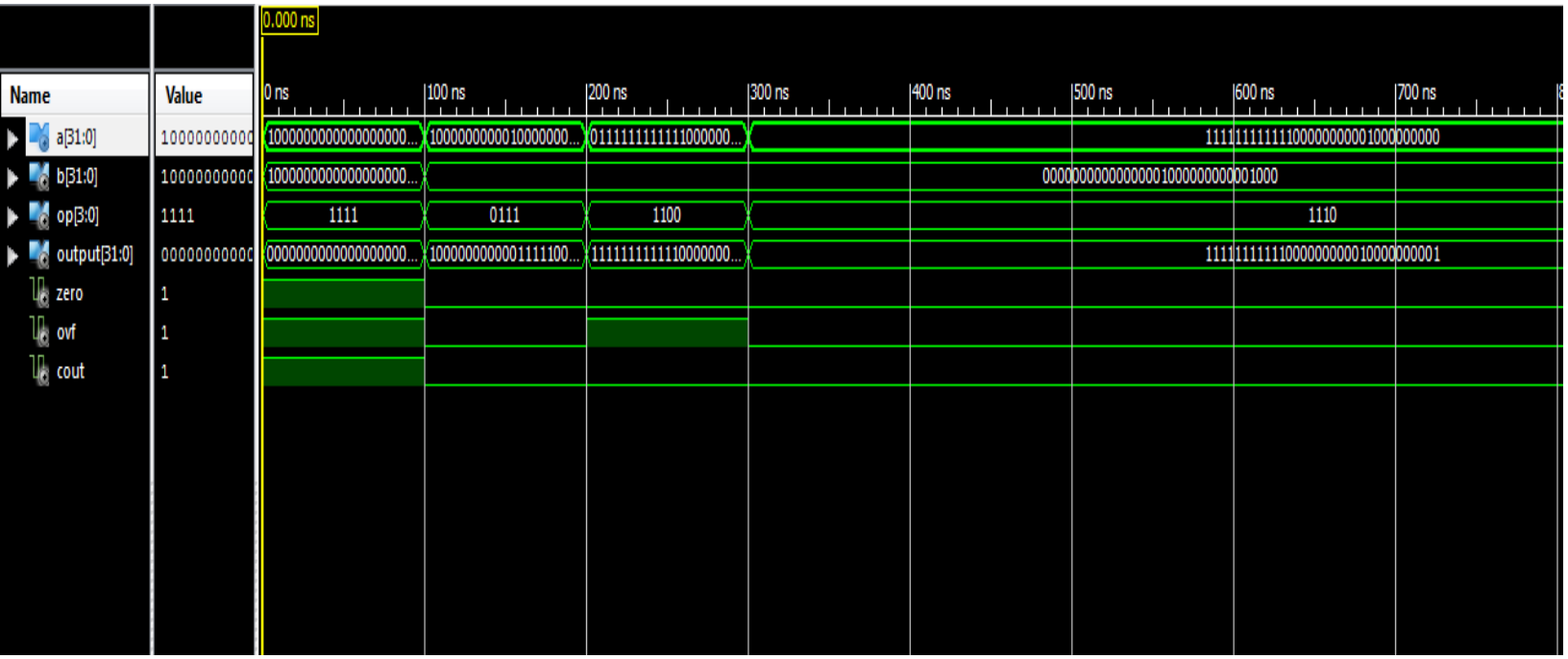
Ακολουθεί ολοκληρωμένος ο κώδικας της ALU:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use ieee.std_logic_arith.all;
entity alu_fk is
 Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
        B : in  STD_LOGIC_VECTOR (31 downto 0);
        op : in  STD_LOGIC_VECTOR (3 downto 0);
        Output : out  STD_LOGIC_VECTOR (31 downto 0);
        zero : out  STD_LOGIC;
        ovf : out  STD_LOGIC;
        Cout : out  STD_LOGIC);
end alu_fk;

architecture Behavioral of alu_fk is

signal b_minus : std_logic_vector (31 downto 0 );
signal temp : std_logic_vector (31 downto 0 );

begin

b_minus <= -B;
WITH op SELECT
        temp <=         (A+B)                   when "1111",  --pros8esi
                        A+b_minus               when "0111",  --afairesi
                        A and B                 when "0011",  --logiko kai
                        not A                   when "0001" , --!A
                        A or B                  when "0000",  --logiko H'
                        '0' & A(31 downto 1)    when "1000",  --olis8isi deksia kata 1 8esi
                        A(30 downto 0) & '0'    when "1100",  --olis8isi aristera kata 1 8esi
                        A(30 downto 0) & A(31)  when "1110",--rotate aristera kata 1 8esi
                        A(0) & A(30 downto 0)   when "0101",--kukliko rotate deksia kata 1 8esi
                        temp WHEN OTHERS;
        Output <= temp; --apotelesma

        zero <= '1' WHEN temp=0 ELSE '0';
        --auta kalo 8a itan na trexoun mono se add kai sub!!!
        WITH A(31) XOR B(31) select
                ovf<=   A(31) xor temp(31)    when '0',
                        '0'                   when others;

        WITH A(31) XOR B(31) select
                Cout<= A(31)            when '0',
                       not temp(31)     when others;

end Behavioral;
```

Καριμπίδης Διονύσης **A.M. : 2011030009**
Χριστοδούλου Θεόφιλος **A.M. : 2011030010**

ALU_Test

Παρακάτω βλέπουμε μία δοκιμαστική λειτουργία της ALU που δημιουργήσαμε:



Στην εικόνα παρατηρούμε με χρονική σειρά:

- Την εκτέλεση μιας **πρόσθεσης** μεταξύ των
  A="10000000000000000000000000000000"
  B="10000000000000000000000000000000"
  από την οποία ακριβώς όπως περιμέναμε προκύπτει αποτέλεσμα
  "00000000000000000000000000000000" , ενώ ενεργοποιούνται και τα σήματα **Ovf**, **Cout** και **Zero**.

- Την εκτέλεση μιας **αφαίρεσης** μεταξύ των
  A="10000000001000000000001000000000"
  B="00000000000000001000000000001000"
  από την οποία προκύπτει επίσης το σωστό αποτέλεσμα

- Την εκτέλεση ενός **Shift Left** του τελεστή
  A=*"01111111111110000000000100000000"* *με σωστό αποτέλεσμα*
  "11111111111100000000001000000000"

- Την εκτέλεση ενός **Rotate Left** του τελεστή
  A=*"11111111111100000000001000000000"* *με σωστό αποτέλεσμα*
  "11111111111000000000010000000001"

Και η ALU μας δουλεύει εξαιρετικά!

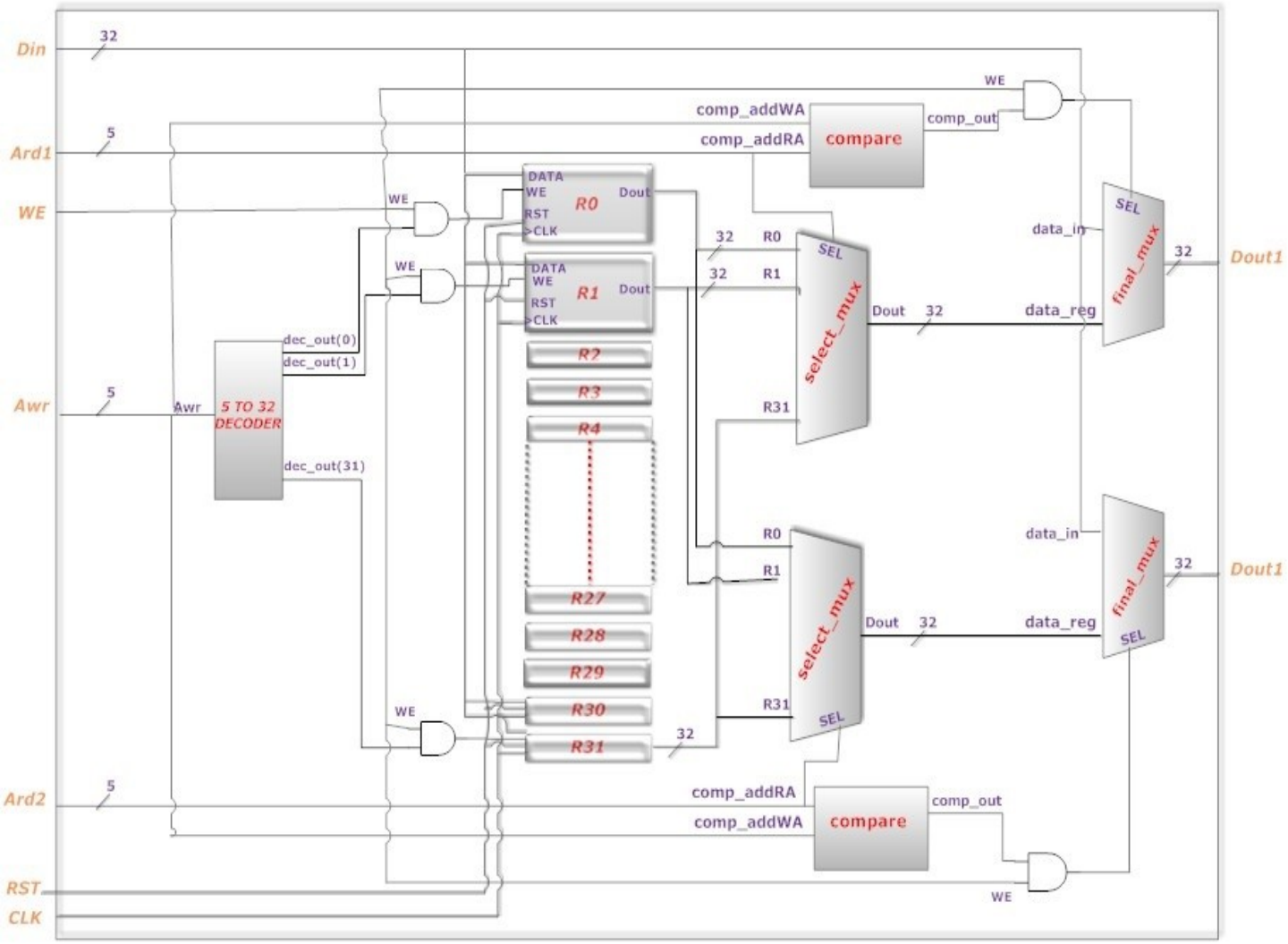
Καριμπίδης Διονύσης **Α.Μ. : 2011030009**
Χριστοδούλου Θεόφιλος **Α.Μ. : 2011030010**

Μάρτιος 2014

2) Σχεδίαση του Αρχείου Καταχωρητών (Register File)

Αρχικά,υλοποιήσαμε έναν 32-bit register ο οποίος περνάει στην έξοδο τα δεδομένα που έχει στην είσοδο όταν το σήμα we = '1'. Με άλλα λόγια, ένα σύγχρονο δομικό στοιχείο που αντιστοιχεί σε μνήμη 32 bit, στην οποία πραγματοποιούμε συνεχώς ανάγνωση.
Στο τέλος της εργαστηριακής άσκησης, προσθέσαμε και το σήμα RST στον register αυτόν ώστε να μπορούμε να αρχικοποιούμε τους καταχωρητές, κάνοντας το σύστημα πιο αξιόπιστο.

Στη συνέχεια συνδέσαμε 32 τέτοιους register όπως περιγράψαμε παραπάνω και δημιουργήσαμε ένα αρχείο καταχωρητών που υλοποιεί μια μνήμη 32 θέσεων των 32 bit. Για την υλοποίηση του register file χρειάστηκαν επίσης κάποιοι πολυπλέκτες, ένα decoder module και δυο compare modules.
Η διασύνδεσή τους γίνεται σαφής στο παρακάτω block diagram:



Ανάλυση Υλοποίησης:

- **Είσοδοι**:
    - Καθολικά σήματα RST και CLK με τις γνωστές λειτουργίες
    - Ard1, Ard2 διευθύνσεις(αριθμός καταχωρητή) ανάγνωσης(5 bit)
    - Arw διεύθυνση εγγραφής(5 bit)
    - WE ενεργοποίηση εγγραφής
    - Din δεδομένα προς εγγραφή(32 bit)

- **Έξοδοι:**
    - Dout1, Dout2 δεδομένα που βρίσκονται στη δεδομένη θέση ανάγνωσης (32 bit)

- ***5 TO 32 DECODER:***

  παίρνει ως είσοδο την διεύθυνση εγγραφής που είναι μια 5bit ποσότητα ώστε να μπορέσει να αναπαραστήσει και τους 32 καταχωρητές και έχει ως έξοδο ένα σήμα 32bit όπου έχει σε όλα του τα bit '0' εκτός από 1bit που είναι '1' και υποδηλώνει τον καταχωρητή εγγραφής.

- Η **έξοδος του *DECODER*** και το σήμα WE(write enable) γίνονται οι είσοδοι σε μια *AND η έξοδος της οποίας καταλήγει στο* WE του κάθε καταχωρητή. Με αυτόν τον τρόπο όταν ενεργοποιηθεί το WE='1' τα δεδομένα θα γραφούν μόνο σε εκείνον τον καταχωρητή που η έξοδος της AND έβγαλε '1'.

- Όλες οι έξοδοι των καταχωρητών οδηγούνται σε 2 ίδιους πολυπλέκτες (***select_mux***) οι οποίου έχουν στην είσοδό τους τα 32 σήματα των 32bit, σήμα ελέγχου την διεύθυνση ανάγνωσης και έξοδο το 32bit σήμα που βρίσκεται στην κατάλληλη διεύθυνση. Ο λόγος που έχουμε 2 τέτοιους πολυπλέκτες είναι για να μπορούμε να διαβάζουμε τα δεδομένα από 2 καταχωρητές ταυτόχρονα.

- Επειδή όμως μπορεί να τύχει η διεύθυνση ανάγνωσης και η διεύθυνση εγγραφής να είναι ίδιες χρησιμοποιούμε ένα ***compare module*** ώστε να το ελέγξουμε αυτό. Αν λοιπόν είναι όντως κοινές το *compare module* βγάζει ως έξοδο '1'.

- Η **έξοδος του *compare*** οδηγείται σε μια *AND* που έχει ως είσοδο και το σήμα WE. Η έξοδος αυτής της AND αποτελεί το control του τελευταίου πολυπλέκτη (final_mux).Έτσι εάν αν η έξοδος του *compare είναι '1'* και το σήμα WE είναι ενεργό, τότε η AND βγάζει έξοδο '1'.

- Ο ***final_mux*** είναι αυτός που αποφασίζει αν στην έξοδο θα εμφανιστούν τα δεδομένα του καταχωρητή ή της εισόδου που δώσαμε. Αυτό ελέγχεται όπως αναφέραμε και παραπάνω απο το σήμα εξόδου της AND που μας δείχνει αν είναι κοινές οι διευθύνσεις εγγραφής και ανάγνωσης και το WE='1'. Τότε βγάζουμε στην έξοδο τα δεδομένα που εισάγαμε(αυτά που μόλις έγιναν εγγραφή-Din),αλλιώς τα δεδομένα του καταχωρητή.

Τέλος,υλοποιήσαμε το top_module κάνοντας τα κατάλληλα port maps και χρησιμοποιώντας τα εσωτερικά signals που μας χρειάστηκαν.

Ακολουθούν οι κώδικες των προαναφερθέντων components για την υλοποίηση του Register File.

Register

```
entity reg is
   Port ( DATA : in  STD_LOGIC_VECTOR (31 downto 0);
          WE : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          Dout : out  STD_LOGIC_VECTOR (31 downto 0));
end reg;

architecture Behavioral of reg is

signal tmp : STD_LOGIC_VECTOR (31 downto 0);

begin

Process(CLK)
begin
        if (CLK'EVENT AND CLK='1') THEN
                if(we='1') then
                        tmp<= DATA;
                ELSE
                        tmp <= tmp;
                end if;
        END IF;
end process;
        Dout <=tmp;
end Behavioral;
```

5 to 32 decoder

```
entity decoder_5_32 is
   Port ( Awr : in  STD_LOGIC_VECTOR (4 downto 0);
          dec_output : out  STD_LOGIC_VECTOR (31 downto 0));
end decoder_5_32;

architecture Behavioral of decoder_5_32 is

begin
with Awr select
          dec_output <=                "00000000000000000000000000000000" when "00000",
                                       "00000000000000000000000000000010" when "00001",
                                       "00000000000000000000000000000100" when "00010",
                                       "00000000000000000000000000001000" when "00011",
                                       "00000000000000000000000000010000" when "00100",
                                       "00000000000000000000000000100000" when "00101",
                                       "00000000000000000000000001000000" when "00110",
                                       "00000000000000000000000010000000" when "00111",
                                       "00000000000000000000000100000000" when "01000",
                                       "00000000000000000000001000000000" when "01001",
                                       "00000000000000000000010000000000" when "01010",
                                       "00000000000000000000100000000000" when "01011",
                                       "00000000000000000001000000000000" when "01100",
                                       "00000000000000000010000000000000" when "01101",
                                       "00000000000000000100000000000000" when "01110",
                                       "00000000000000001000000000000000" when "01111",
                                       "00000000000000010000000000000000" when "10000",
                                       "00000000000000100000000000000000" when "10001",
                                       "00000000000001000000000000000000" when "10010",
                                       "00000000000010000000000000000000" when "10011",
                                       "00000000000100000000000000000000" when "10100",
                                       "00000000001000000000000000000000" when "10101",
                                       "00000000010000000000000000000000" when "10110",
                                       "00000000100000000000000000000000" when "10111",
                                       "00000001000000000000000000000000" when "11000",
                                       "00000010000000000000000000000000" when "11001",
                                       "00000100000000000000000000000000" when "11010",
                                       "00001000000000000000000000000000" when "11011",
                                       "00010000000000000000000000000000" when "11100",
                                       "00100000000000000000000000000000" when "11101",
                                       "01000000000000000000000000000000" when "11110",
                                       "10000000000000000000000000000000" when others;

end Behavioral;
```

Καριμπίδης Διονύσης **A.M. : 2011030009**
Χριστοδούλου Θεόφιλος **A.M. : 2011030010**

Select_mux

```
entity select_mux is
    Port ( R0 : in  STD_LOGIC_VECTOR (31 downto 0);
           R1 : in  STD_LOGIC_VECTOR (31 downto 0);
           R2 : in  STD_LOGIC_VECTOR (31 downto 0);
           R3 : in  STD_LOGIC_VECTOR (31 downto 0);
           R4 : in  STD_LOGIC_VECTOR (31 downto 0);
           R5 : in  STD_LOGIC_VECTOR (31 downto 0);
           R6 : in  STD_LOGIC_VECTOR (31 downto 0);
           R7 : in  STD_LOGIC_VECTOR (31 downto 0);
           R8 : in  STD_LOGIC_VECTOR (31 downto 0);
           R9 : in  STD_LOGIC_VECTOR (31 downto 0);
           R10 : in  STD_LOGIC_VECTOR (31 downto 0);
           R11 : in  STD_LOGIC_VECTOR (31 downto 0);
           R12 : in  STD_LOGIC_VECTOR (31 downto 0);
           R13 : in  STD_LOGIC_VECTOR (31 downto 0);
           R14 : in  STD_LOGIC_VECTOR (31 downto 0);
           R15 : in  STD_LOGIC_VECTOR (31 downto 0);
           R16 : in  STD_LOGIC_VECTOR (31 downto 0);
           R17 : in  STD_LOGIC_VECTOR (31 downto 0);
           R18 : in  STD_LOGIC_VECTOR (31 downto 0);
           R19 : in  STD_LOGIC_VECTOR (31 downto 0);
           R20 : in  STD_LOGIC_VECTOR (31 downto 0);
           R21 : in  STD_LOGIC_VECTOR (31 downto 0);
           R22 : in  STD_LOGIC_VECTOR (31 downto 0);
           R23 : in  STD_LOGIC_VECTOR (31 downto 0);
           R24 : in  STD_LOGIC_VECTOR (31 downto 0);
           R25 : in  STD_LOGIC_VECTOR (31 downto 0);
           R26 : in  STD_LOGIC_VECTOR (31 downto 0);
           R27 : in  STD_LOGIC_VECTOR (31 downto 0);
           R28 : in  STD_LOGIC_VECTOR (31 downto 0);
           R29 : in  STD_LOGIC_VECTOR (31 downto 0);
           R30 : in  STD_LOGIC_VECTOR (31 downto 0);
           R31 : in  STD_LOGIC_VECTOR (31 downto 0);
           SEL : in  STD_LOGIC_VECTOR (4 downto 0);
           Dout : out  STD_LOGIC_VECTOR (31 downto 0));
end select_mux;


architecture Behavioral of select_mux is

begin
```

```
with SEL select
        Dout <= R0 when "00000",
                        R1 when "00001",
                        R2 when "00010",
                        R3 when "00011",
                        R4 when "00100",
                        R5 when "00101",
                        R6 when "00110",
                        R7 when "00111",
                        R8 when "01000",
                        R9 when "01001",
                        R10 when "01010",
                        R11 when "01011",
                        R12 when "01100",
                        R13 when "01101",
                        R14 when "01110",
                        R15 when "01111",
                        R16 when "10000",
                        R17 when "10001",
                        R18 when "10010",
                        R19 when "10011",
                        R20 when "10100",
                        R21 when "10101",
                        R22 when "10110",
                        R23 when "10111",
                        R24 when "11000",
                        R25 when "11001",
                        R26 when "11010",
                        R27 when "11011",
                        R28 when "11100",
                        R29 when "11101",
                        R30 when "11110",
                        R31 when "11111",
                        Dout when others;

end Behavioral;
```

Compare

```
entity compare is
   Port ( comp_addRA : in  STD_LOGIC_VECTOR (4 downto 0);
          comp_addWA : in  STD_LOGIC_VECTOR (4 downto 0);
          comp_out : out  STD_LOGIC);
end compare;

architecture Behavioral of compare is

begin

comp_out <= '1' when comp_addRA = comp_addWA else
                                    '0';

end Behavioral;
```

Final_mux

```
entity final_mux is
   Port ( sel : in  STD_LOGIC;
          data_reg : in  STD_LOGIC_VECTOR (31 downto 0);
          data_in : in  STD_LOGIC_VECTOR (31 downto 0);
          output_addr : out  STD_LOGIC_VECTOR (31 downto 0));
end final_mux;

architecture Behavioral of final_mux is

begin

output_addr <= data_reg when sel='0' else
                                        data_in;

end Behavioral;
```

Top level

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RF_top_omikronpsi420 is
    Port ( Ard1 : in  STD_LOGIC_VECTOR (4 downto 0);
           Ard2 : in  STD_LOGIC_VECTOR (4 downto 0);
           Awr : in  STD_LOGIC_VECTOR (4 downto 0);
           Din : in  STD_LOGIC_VECTOR (31 downto 0);
           WE : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Dout1 : out  STD_LOGIC_VECTOR (31 downto 0);
           Dout2 : out  STD_LOGIC_VECTOR (31 downto 0));
end RF_top_omikronpsi420;

architecture Behavioral of RF_top_omikronpsi420 is

component decoder_5_32 is
    Port ( Awr : in  STD_LOGIC_VECTOR (4 downto 0);
           dec_output : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

component reg is
    Port ( DATA : in  STD_LOGIC_VECTOR (31 downto 0);
           WE : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
           Dout : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

component select_mux is
    Port ( R0 : in  STD_LOGIC_VECTOR (31 downto 0);
           R1 : in  STD_LOGIC_VECTOR (31 downto 0);
           R2 : in  STD_LOGIC_VECTOR (31 downto 0);
           R3 : in  STD_LOGIC_VECTOR (31 downto 0);
           R4 : in  STD_LOGIC_VECTOR (31 downto 0);
           R5 : in  STD_LOGIC_VECTOR (31 downto 0);
           R6 : in  STD_LOGIC_VECTOR (31 downto 0);
           R7 : in  STD_LOGIC_VECTOR (31 downto 0);
           R8 : in  STD_LOGIC_VECTOR (31 downto 0);
           R9 : in  STD_LOGIC_VECTOR (31 downto 0);
           R10 : in  STD_LOGIC_VECTOR (31 downto 0);
           R11 : in  STD_LOGIC_VECTOR (31 downto 0);
           R12 : in  STD_LOGIC_VECTOR (31 downto 0);
           R13 : in  STD_LOGIC_VECTOR (31 downto 0);
           R14 : in  STD_LOGIC_VECTOR (31 downto 0);
           R15 : in  STD_LOGIC_VECTOR (31 downto 0);
           R16 : in  STD_LOGIC_VECTOR (31 downto 0);
           R17 : in  STD_LOGIC_VECTOR (31 downto 0);
```

```
        R18 : in  STD_LOGIC_VECTOR (31 downto 0);
        R19 : in  STD_LOGIC_VECTOR (31 downto 0);
        R20 : in  STD_LOGIC_VECTOR (31 downto 0);
        R21 : in  STD_LOGIC_VECTOR (31 downto 0);
        R22 : in  STD_LOGIC_VECTOR (31 downto 0);
        R23 : in  STD_LOGIC_VECTOR (31 downto 0);
        R24 : in  STD_LOGIC_VECTOR (31 downto 0);
        R25 : in  STD_LOGIC_VECTOR (31 downto 0);
        R26 : in  STD_LOGIC_VECTOR (31 downto 0);
        R27 : in  STD_LOGIC_VECTOR (31 downto 0);
        R28 : in  STD_LOGIC_VECTOR (31 downto 0);
        R29 : in  STD_LOGIC_VECTOR (31 downto 0);
        R30 : in  STD_LOGIC_VECTOR (31 downto 0);
        R31 : in  STD_LOGIC_VECTOR (31 downto 0);
        SEL : in  STD_LOGIC_VECTOR (4 downto 0);
        Dout : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

component compare is
    Port ( comp_addRA : in  STD_LOGIC_VECTOR (4 downto 0);
           comp_addWA : in  STD_LOGIC_VECTOR (4 downto 0);
           comp_out : out  STD_LOGIC);
end component;

component final_mux is
    Port ( sel : in  STD_LOGIC;
           data_reg : in  STD_LOGIC_VECTOR (31 downto 0);
           data_in : in  STD_LOGIC_VECTOR (31 downto 0);
           output_addr : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

signal dec_out, sel_mux_out, reg_out1, reg_out2, reg_out3, reg_out4, reg_out5, reg_out6, reg_out7,
reg_out8, reg_out9, reg_out10, reg_out11, reg_out12, reg_out13, reg_out14, reg_out15, reg_out16,
reg_out17, reg_out18, reg_out19, reg_out20, reg_out21, reg_out22, reg_out23, reg_out24,
reg_out25, reg_out26, reg_out27, reg_out28, reg_out29, reg_out30, reg_out31, reg_out32 :
STD_LOGIC_VECTOR (31 downto 0);
signal comp_out, sel_final, we_reg_sig1, we_reg_sig2, we_reg_sig3, we_reg_sig4, we_reg_sig5,
we_reg_sig6, we_reg_sig7, we_reg_sig8, we_reg_sig9, we_reg_sig10, we_reg_sig11, we_reg_sig12,
we_reg_sig13, we_reg_sig14, we_reg_sig15, we_reg_sig16, we_reg_sig17, we_reg_sig18,
we_reg_sig19, we_reg_sig20, we_reg_sig21, we_reg_sig22, we_reg_sig23, we_reg_sig24,
we_reg_sig25, we_reg_sig26, we_reg_sig27, we_reg_sig28, we_reg_sig29, we_reg_sig30,
we_reg_sig31, we_reg_sig32 : STD_LOGIC;
signal final_mux_out : STD_LOGIC_VECTOR(4 downto 0);
```

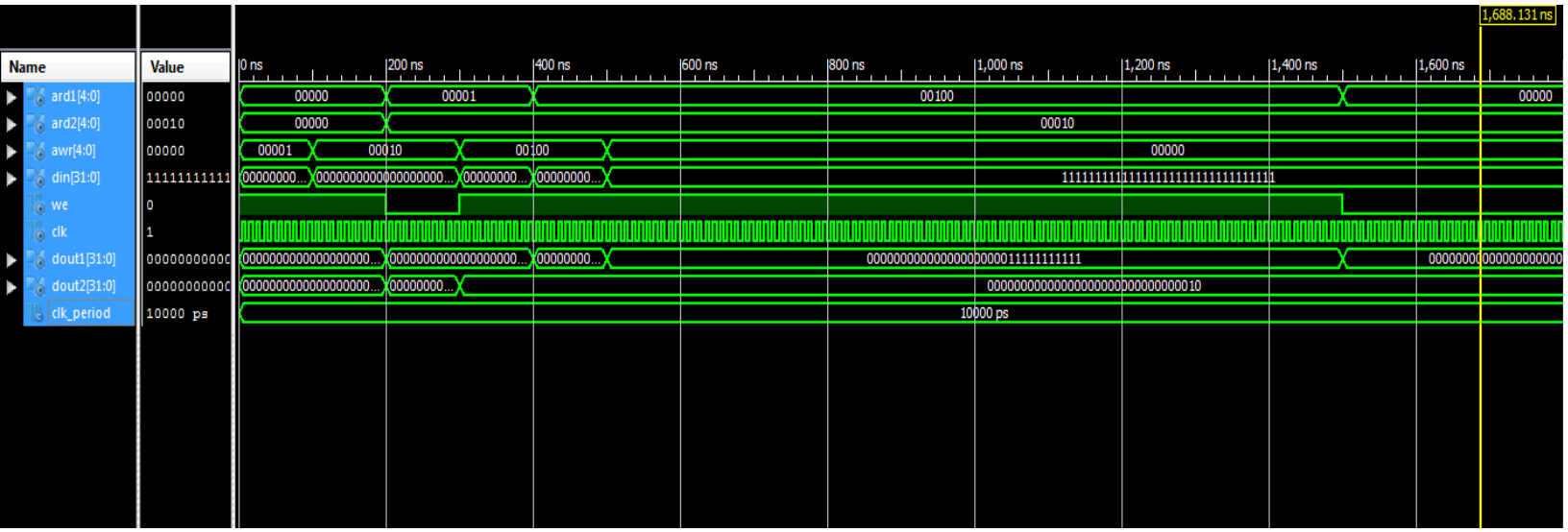
```
begin
flag1:  decoder_5_32 port map(      Awr => Awr,
                                    dec_output =>       dec_out);

reg1:   reg port map ( DATA => Din,
                                                WE => we_reg_sig1,
                                                CLK => CLK,
                                                Dout => reg_out1);

reg2:   reg port map ( DATA => Din,
                                                WE => we_reg_sig2,
                                                CLK => CLK,
                                                Dout => reg_out2);

reg3:   reg port map ( DATA => Din,
                                                WE => we_reg_sig3,
                                                CLK => CLK,
                                                Dout => reg_out3);

reg4:   reg port map ( DATA => Din,
                                                WE => we_reg_sig4,
                                                CLK => CLK,
                                                Dout => reg_out4);

reg5:   reg port map ( DATA => Din,
                                                WE => we_reg_sig5,
                                                CLK => CLK,
                                                Dout => reg_out5);

reg6:   reg port map ( DATA => Din,
                                                WE => we_reg_sig6,
                                                CLK => CLK,
                                                Dout => reg_out6);

reg7:   reg port map ( DATA => Din,
                                                WE => we_reg_sig7,
                                                CLK => CLK,
                                                Dout => reg_out7);

reg8:   reg port map ( DATA => Din,
                                                WE => we_reg_sig8,
                                                CLK => CLK,
                                                Dout => reg_out8);

reg9:   reg port map ( DATA => Din,
                                                WE => we_reg_sig9,
                                                CLK => CLK,
                                                Dout => reg_out9);
```

```
reg10: reg port map ( DATA => Din,
                      WE => we_reg_sig10,
                      CLK => CLK,
                      Dout => reg_out10);

reg11: reg port map ( DATA => Din,
                      WE => we_reg_sig11,
                      CLK => CLK,
                      Dout => reg_out11);

reg12: reg port map ( DATA => Din,
                      WE => we_reg_sig12,
                      CLK => CLK,
                      Dout => reg_out12);

reg13: reg port map ( DATA => Din,
                      WE => we_reg_sig13,
                      CLK => CLK,
                      Dout => reg_out13);

reg14: reg port map ( DATA => Din,
                      WE => we_reg_sig14,
                      CLK => CLK,
                      Dout => reg_out14);

reg15: reg port map ( DATA => Din,
                      WE => we_reg_sig15,
                      CLK => CLK,
                      Dout => reg_out15);

reg16: reg port map ( DATA => Din,
                      WE => we_reg_sig16,
                      CLK => CLK,
                      Dout => reg_out16);

reg17:  reg port map ( DATA => Din,
                      WE => we_reg_sig17,
                      CLK => CLK,
                      Dout => reg_out17);


reg18: reg port map ( DATA => Din,
                      WE => we_reg_sig18,
                      CLK => CLK,
                      Dout => reg_out18);


reg19: reg port map ( DATA => Din,
                      WE => we_reg_sig19,
```

```
                                        CLK => CLK,
                                        Dout => reg_out19);

reg20: reg port map ( DATA => Din,
                                        WE => we_reg_sig20,
                                        CLK => CLK,
                                        Dout => reg_out20);

reg21: reg port map ( DATA => Din,
                                        WE => we_reg_sig21,
                                        CLK => CLK,
                                        Dout => reg_out21);

reg22: reg port map ( DATA => Din,
                                        WE => we_reg_sig22,
                                        CLK => CLK,
                                        Dout => reg_out22);

reg23: reg port map ( DATA => Din,
                                        WE => we_reg_sig23,
                                        CLK => CLK,
                                        Dout => reg_out23);

reg24: reg port map ( DATA => Din,
                                        WE => we_reg_sig24,
                                        CLK => CLK,
                                        Dout => reg_out24);

reg25: reg port map ( DATA => Din,
                                        WE => we_reg_sig25,
                                        CLK => CLK,
                                        Dout => reg_out25);

reg26: reg port map ( DATA => Din,
                                        WE => we_reg_sig26,
                                        CLK => CLK,
                                        Dout => reg_out26);

reg27: reg port map ( DATA => Din,
                                        WE => we_reg_sig27,
                                        CLK => CLK,
                                        Dout => reg_out27);

end Behavioral;
```

Register_File_Test

Παρακάτω βλέπουμε μια δοκιμαστική λειτουργία του Register File που εν τέλει δημιουργήσαμε.



Στην ουσία, με χρονική σειρά:

-**Γράφουμε** το (32bit-δυαδικό) *'1'* στον πρώτο καταχωρητή **R1**

-**Γράφουμε** το (32bit-δυαδικό) *'2'* στον πρώτο καταχωρητή **R2** και στη συνέχεια

-**Διαβάζουμε** το περιεχόμενο των 2 πρώτων καταχωρητών **R1, R2** και στην έξοδο εμφανίζονται οι τιμές που μόλις πριν εγγράψαμε

-**Γράφουμε** το *"00000000000000000000000000000000"* στον **R4**

-**Γράφουμε** *"00000000000000000000011111111111"* στον **R4 και διαβάζουμε <u>ταυτόχρονα</u>**. Στην έξοδο περνάει ορθά η νέα τιμή του Register.

-**Γράφουμε**, τέλος στον **R0** την τιμή *"11111111111111111111111111111111"* και

-**Διαβάζουμε** τον **R0** επιβεβαιώνοντας πως δεν μπορούμε να γράψουμε τίποτα σε αυτόν, καθώς στην έξοδο εμφανίζεται το "00000000000000000000000000000000".

Η λειτουργία του Register File μας, λοιπόν, είναι <u>εντυπωσιακά καλή, όπως ακριβώς περιμέναμε</u>....!

Καριμπίδης Διονύσης **Α.Μ. : 2011030009**
Χριστοδούλου Θεόφιλος **Α.Μ. : 2011030010**

Μάρτιος 2014