ΟΡΓΑΝΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ
ΗΡΥ 312

ΕΡΓΑΣΤΗΡΙΟ 5
Αναφορά Εργαστηριακής Άσκησης

Ομάδα Εργασίας **LAB31220188**

α/α	Α.Μ.	Ονοματεπώνυμο
1	2011030010	Χριστοδουλου Θεοφιλος
2	2011030009	ΚΑΡΙΜΠΙΔΗΣ ΔΙΟΝΥΣΗΣ

Εισαγωγούλα..

Στην 5η και τελευταία Εργαστηριακή Άσκηση κληθήκαμε να τροποποιήσουμε τον επεξεργαστή πολλαπλών κύκλων,που δημιουργήσαμε στα προηγούμενα εργαστήρια, ώστε να πραγματοποιεί τις εντολές γρηγορότερα,μέσω pipeline. Στην ουσία, η κάθε εντολή μας πλέον διαρκούσε 5 κύκλους,κάθε ένας από τους οποίους “αναφερόταν” σε ένα εκ των **IF_stage**(διάβασμα εντολής),**DEC_stage**(read mode), **ALU_stage**, **MEM_stage**, **DEC_stage**(write mode).

Σε κάθε κύκλο του ρολογιού, πια, διαβάζεται μια νέα εντολή.

Το control_module παράγει τα κατάλληλα σήματα και τα περνά σε μια σειρά από 4 καταχωρητές (reg12,reg11,reg6,reg5) οι οποίοι “μεταφέρουν” την πληροφορία με την κατάλληλη καθυστέρηση στο κατάλληλο module. Η έξοδος του control είναι ένα 14bit σήμα το οποίο κατανέμεται ως εξής:

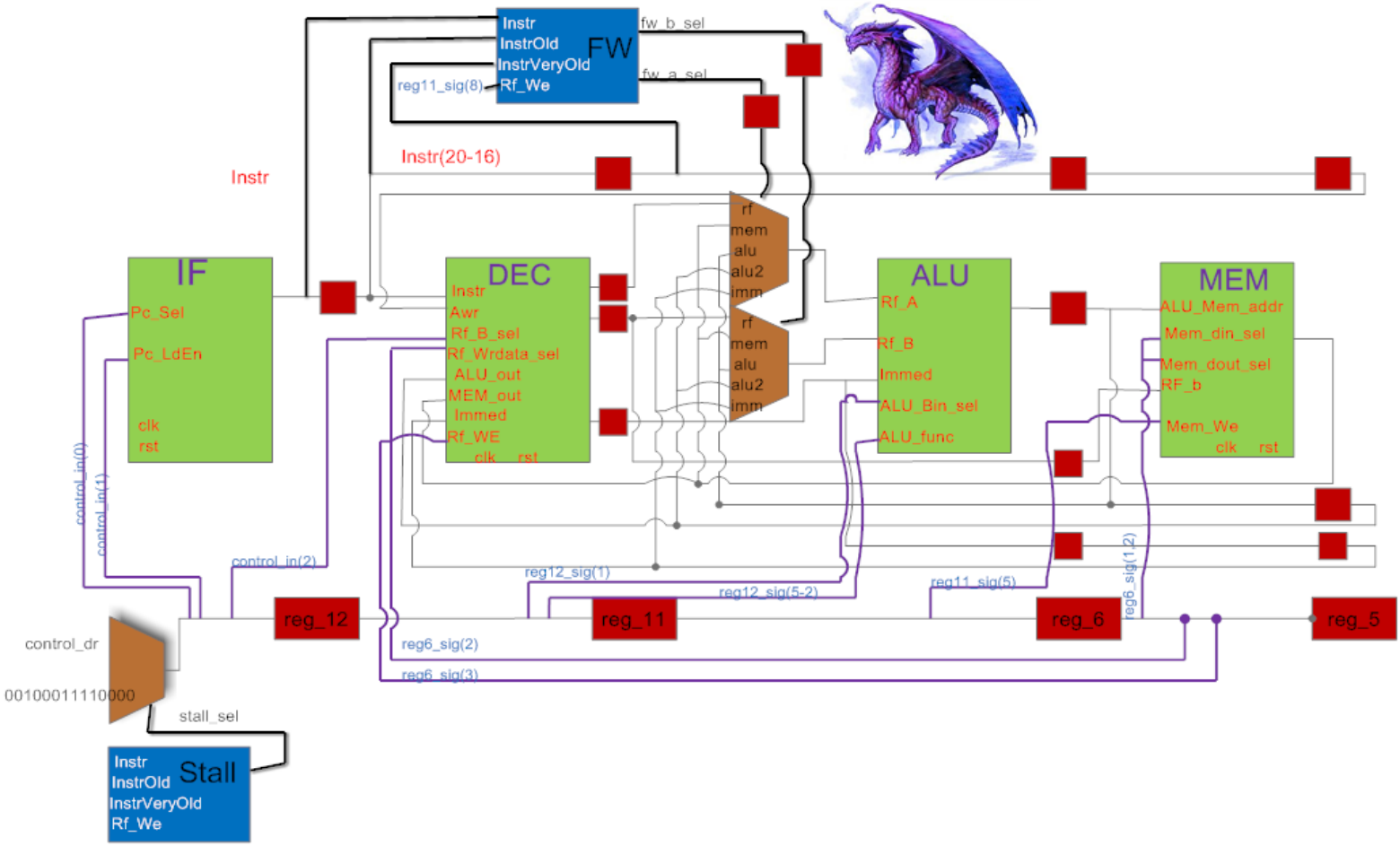
control_out(0) → pc_sel,
control_out(1) → pc_loadEnable,
control_out(2) → rf_b_sel,
control_out(3) → alu_b_sel,
control_out(7 downto 4) → alu_func,
control_out(8) → mem_WriteEnable,
control_out(10 downto 9) → πάντα '0' διότι αναφέρονται στους πολυπλέκτες για lb και sb και δεν τα χρησιμοποιούμε
control_out(11) → rt_wrEnable,
control_out(13 downto 12) → rf_wrData_sel

Η λογική με την οποια λειτουργούμε είναι να συνδέουμε τα σήματα που χρησιμοποιούμε σε κάθε στάδιο και τα υπόλοιπα να τα μεταβιβάζουμε στους επόμενους καταχωρητές του control.

Επιπλέον καταχωρητές χρειάστηκαν για την αντίστοιχη “μεταφορά” της πληροφορίας που παράγεται από κάθε module, καθυστερώντας την όσους κύκλους χρειάζεται ώστε να φτάσει στο κατάλληλο module που θα χρησιμοποιηθεί τη σωστή στιγμή.

Στην ουσία, blocks καταχωρητών παρεμβάλλονται μεταξύ των
IF_stage-DEC_stage
DEC_stage-ALU_stage
ALU_stage-MEM_stage
MEM_stage-DEC_stage

Τα παραπάνω φαίνονται πιο αναλυτικά στο ακόλουθο σχεδιάγραμμα:



**Το stall module παίρνει ακριβώς τις ίδιες εισόδους με το Forward module*

DATA HAZARDS

Το Control δίνει στο datapath σε κάθε κύκλο τα κατάλληλα σήματα για την πλήρη εκτέλεση της νέας εντολής που μόλις διαβάστηκε. Περνώντας την εντολή αυτή με τη σειρά από τους καταχωρητές reg12,reg11,reg6,reg5 καταφέρνουμε να δημιουργήσουμε τις 5 “καταστάσεις” που χρειάζονται για την εκτέλεση, χωρίς το control να είναι FSM.

Ωστόσο, το Pipeline οδήγησε σε πιθανά data hazards, που προκύπτουν όταν μία εντολή**(β)** πάει να διαβάσει (*rs/rt*) κάτι το οποίο γράφεται(*rd*) μία ή δύο εντολές παραπάνω**(α)**. Στην περίπτωση αυτή, η εντολή **β** θα διαβάσει λάθος δεδομένα, αφού η **α** ολοκληρώνεται γράφοντας πίσω στην RF δύο κύκλους αργότερα ή έναν αντίστοιχα.

FORWARD

Για την επίλυση των data hazards δημιουργήσαμε ένα forward module το οποίο ελέγχει εάν η διεύθυνση rs ή rt που πάμε να διαβάσουμε είναι ίδια με την διεύθυνση που γράφει η προηγούμενη ή η πιο προηγούμενη εντολή. Σε αυτή τη περίπτωση στέλνουμε το κατάλληλο σήμα (καθυστερημένο κατά έναν κύκλο)σε έναν πολυπλέκτη ο οποίος θα διαλέξει ποια τιμή θα κάνει forward. Αντίστοιχα δημιουργούμε και έναν 2ο τέτοιο πολυπλέκτη στην 2η είσοδο της alu.
Για του λόγου το αληθές..Ο κώδικας(εμφανίζεται μόνο για το rf_a,απολύτως αντίστοιχα για rf_b):

```
signal tmp1, tmp2 : std_logic_vector(2 downto 0):= "000";
begin

	process

		begin

			WAIT UNTIL clk'EVENT AND clk='1';

			if (rf_wrEn = '1') then

------------rf_a--------------------------

				if (Instrcode(25 downto 21)=instrOld(20 downto 16) and Instrcode(25 downto 21)/="00000") then

					if (instrOld(31 downto 26)="110000") then

						tmp1 <= "011";

--					elsif(instrOld(31 downto 26)="100000") then

--						tmp1 <= "111";

					else

						tmp1 <= "000";

					end if;

				elsif (Instrcode(25 downto 21)=instrVeryOld(20 downto 16) and Instrcode(25 downto 21)/="00000")
then

					if (instrVeryOld(31 downto 26)="110000") then

						tmp1 <= "011";

					elsif(instrVeryOld(31 downto 26)="100000") then

						tmp1 <= "111";

					else

						tmp1 <= "010";

					end if;

				else

					tmp1 <= "000";

				end if;
```

STALL

Στην περίπτωση που έχουμε load εντολές όπου το δεδομένο για εγραφή στην rf έχει δημιουργηθεί στον 4ο κύκλο θα πρέπει να κάνουμε stall, δηλαδή να “αργήσουμε” την επόμενη εντολή. Ελέγχουμε λοιπόν στο stall_module την περίπτωση αυτή και στέλνουμε το κατάλληλο σήμα σε έναν πολυπλέκτη που έχουμε δημιουργήσει, ο οποίος δέχεται ως είσοδο το κανονικό σήμα απο το control ή το σήμα 00100011110000 (μια nop εντολή στην ουσία) ώστε να κάνει stall, δηλαδή να καθυστερήσει την εκτέλεση της εντολής κατά1 κύκλο ώστε να προλάβουν πρώτα να γραφτούν τα δεδομένα.
Ακολουθεί ο κώδικας του stall module (και πάλι μόνο για το rf_a..)

```
signal tmp : std_logic;

begin
 process
                begin
                        WAIT UNTIL clk'EVENT AND clk='1';
                        if (rf_wrEn = '1') then

                                ----------rf_a-------------------------
                                if (Instrcode(25 downto 21)=instrOld(20 downto 16) and
Instrcode(25 downto 21)/="00000000000000000000000000000000") then
                                        if (instrOld(31 downto 26)="001111") then
                                                tmp <= '1';
                                        else
                                                tmp <= '0';
                                        end if;
                                else
                                        tmp <= '0';
                                end if;
```

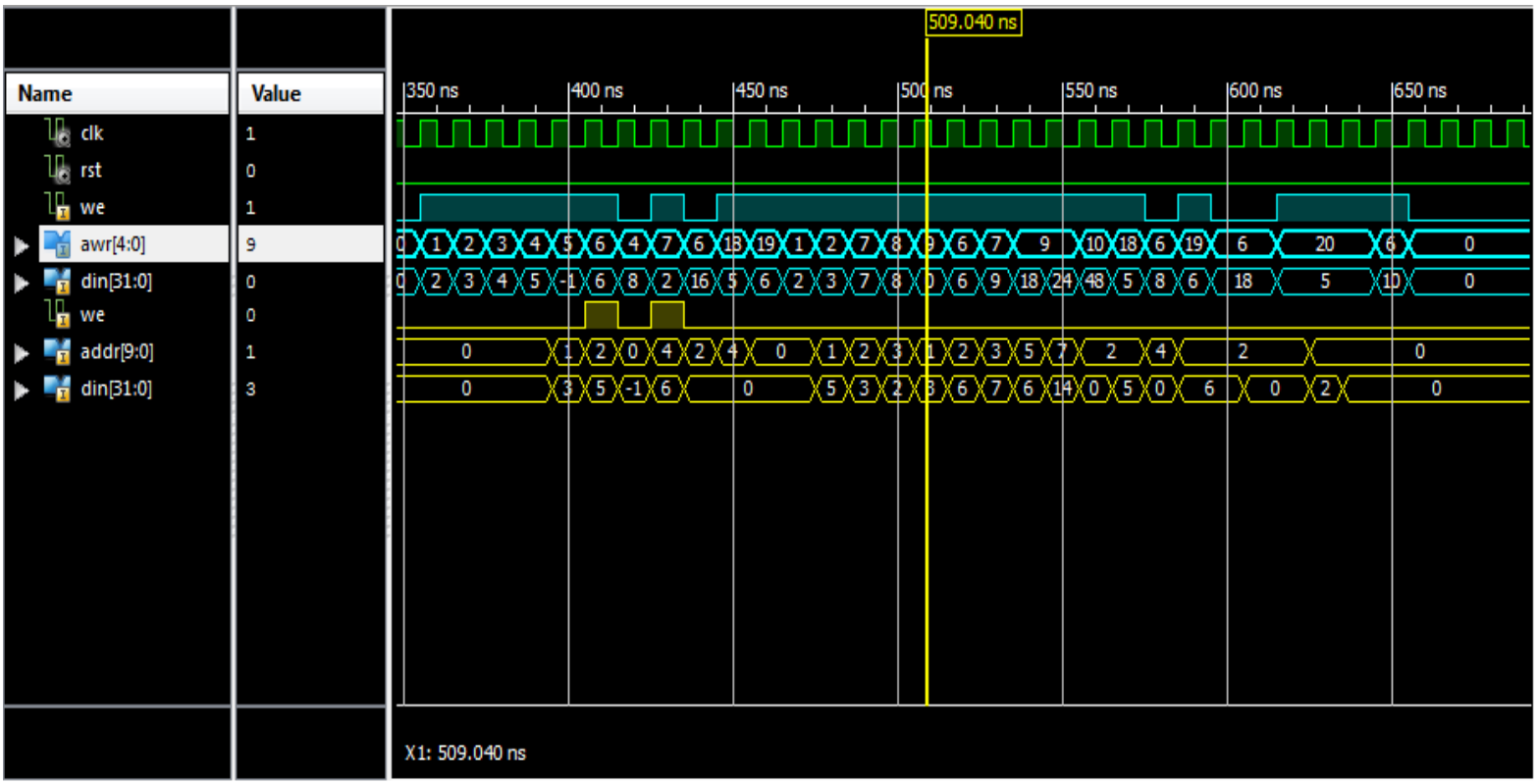
Simulation

Η τελική προσομοίωση του συστήματος ολοκληρώθηκε, πραγματοποιώντας σωστά όλο το πρόγραμμα που δόθηκε προς εκτέλεση για το εργαστήριο αυτό.

Παρακάτω εμφανίζεται ένα snapshot του simulation το οποίο δείχνει

- ✓ Πότε γράφουμε -**WE**
- ✓ Πού γράφουμε -**Awr/Addr**
- ✓ Τι γράφουμε -**Din**

Στην **RF (με γαλάζιο)** και στη μνήμη **MEM (με κίτρινο)**



Χριστοδούλου Θεόφιλος **A.M.: 2011030010**

Καριμπίδης Διονύσης **A.M.: 2011030009**