10/4/2017

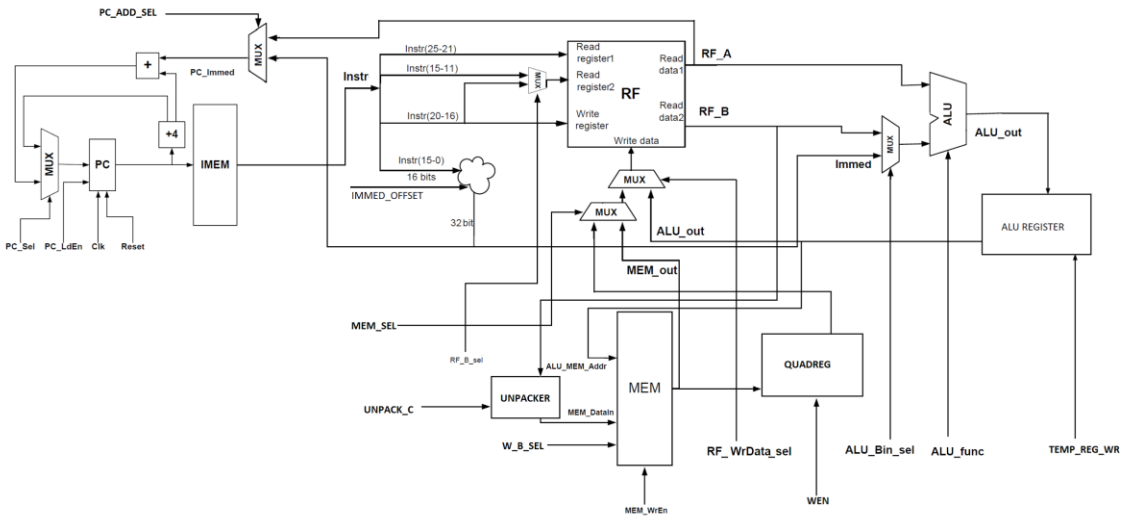
Αναφορά Εργαστηρίου 4

Κωδικός Ομάδας **LAB31231465**

Διαλεκτάκης Γιώργος
Βαϊλάκης Απόστολος Νικόλαος

Προεργασία

Ως προεργασία του 4ου εργαστηρίου μας ζητήθηκε ένα σχηματικό διάγραμμα του **Datapath** που υλοποιεί κάποιες σύνθετες εντολές όπου φαίνονται οι αλλαγές που έχουμε κάνει σε σχέση με τα προηγούμενα εργαστήρια καθώς και της μηχανής πεπερασμένων καταστάσεων του **Control** που ελέγχει το νέο **Datapath.**



Περιγραφή Άσκησης

Σκοπός της 4ης εργαστηριακής άσκησης ήταν η επέκταση της μονάδας Ελέγχου(**Control**) του νέου **Datapath** το οποίο πλέον σχεδιάστηκε για να υλοποιεί σύνθετες εντολές (**cmov, add_MMX_byte, branch_equal_reg_mem, branch_not_equal_reg_mem, byte_pack_mem και byte_unpack_mem**) τις οποίες θα αναλύσουμε παρακάτω. Τελικός σκοπός η ένωση των δύο παραπάνω για την κατασκευή ενός Επεξεργαστή(**Processor**) πολλαπλών κύκλων που υλοποιεί σύνθετες εντολές (**ECHARIS**).

Datapath:

Το **Datapath** του 4ου εργαστηρίου είναι ουσιαστικά το ίδιο με αυτό του 3ου με κάποια πρόσθετα στοιχεία. Πιο συγκεκριμένα δημιουργήσαμε ένα module με όνομα **QUADREG** το οποίο χειρίζεται τις εντολές **branch_not_equal_mem** και **byte_pack_mem** ανάλογα με την είσοδο που θα του δώσει το **Control.** Επίσης, φτιάξαμε ένα module με όνομα **UNPACKER** το οποίο υλοποιεί την εντολή **byte_unpack_mem.** Ακόμα δημιουργήθηκε ένα module μέσα στην **ALU** το οποίο εκτελεί την εντολή **add_MMX_byte** και απλά χρειάζεται τον σωστό Opcode για να λειτουργήσει τον οποίο και παίρνει από το **Control.** Για τις εντολές **cmov** και **branch_equal_reg_mem** δεν δημιουργήθηκαν έξτρα modules αλλά η υλοποίηση έγινε μέσα στο **Control** επειδή θεωρήθηκε ευκολότερο και πιο γρήγορο. Προστέθηκαν ακόμα δύο πολυπλέκτες, ένας στην **IF_STAGE** που αποφασίζει αν θα προστεθεί ο καταχωρητής **rs** ή ο **Immediate** στον **PC** ανάλογα με την εντολή που εκτελούμε (**BREM** ή **BNEM** αντίστοιχα) και ένας στην **MEM_STAGE** που αποφασίζει αν θα γράψουμε στην **Register File** τα δεδομένα της μνήμης ή του QUADREG.

Control:

Η μηχανή πεπερασμένων καταστάσεων που υλοποιεί το Control του νέου Datapath έχει κάποιες έξτρα καταστάσεις σε σχέση με αυτήν του 3ου εργαστηρίου ανάλογα την εντολή που εκτελεί ο επεξεργαστής μας:

Πιο συγκεκριμένα:

-cmov(R-TYPE): Έχουμε την κατάσταση **CMOV** κατά την οποία ελέγχουμε αν ο καταχωρητής **rt** είναι μηδέν και έτσι προχωράμε σε **NOP,** διαφορετικά οδηγούμαστε στην κατάσταση **CMOV_WR** κατά την οποία γράφουμε στην Register File τα δεδομένα του καταχωρητή **rs**.

-add_MMX_byte(R-TYPE): Για την συγκεκριμένη εντολή δεν έχουμε δημιουργήσει έξτρα states. Περνάμε απλά τον κατάλληλο **Opcode** στην **ALU** για να εκτελέσει την σωστή λειτουργία. Την θεωρούμε **R-TYPE** εντολή και επομένως τα αντίστοιχα σήματα θα ενεργοποιηθούν για αυτήν την εντολή όπως και για άλλες **R-TYPE** εντολές.

-branch_equal_reg_mem(I-TYPE): Εδώ έχουμε την κατάσταση **BERM** στην οποία φθάνουμε όταν έρθει το κατάλληλο instruction("**100111**"), την **BERM_LD** κατά την οποία διαβάζουμε από την μνήμη στην διεύθυνση που δείχνει ο **Immediate** και κάνουμε σύγκριση τα δεδομένα της με αυτά του καταχωρητή **rd** και την κατάσταση **BERM_PC** όπου επιλέγουμε τον **PC** δηλαδή αν θα υπάρξει διακλάδωση ή όχι. Στην περίπτωση που υπάρχει διακλάδωση θετουμε στο PC το PC + 4 + RF[Rs].

-branch_not_equal_mem(I-TYPE): Σε αυτή την σύνθετη εντολή έχουμε την κατάσταση

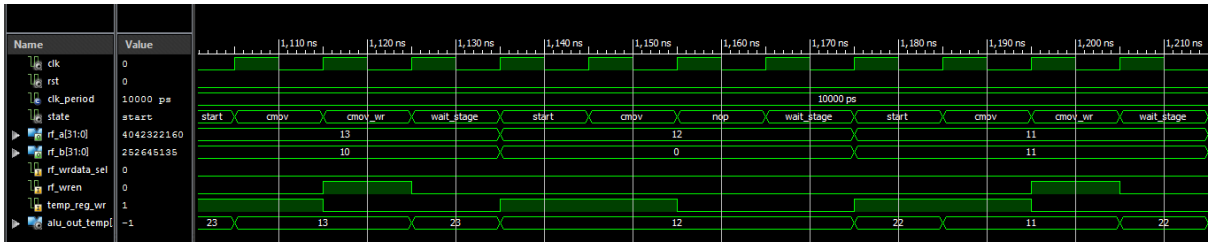
BNEM στην οποία οδηγούμαστε όταν έρθει το κατάλληλο Instruction("**101111**"), συνεχίζουμε με την κατάσταση **BNEM_LD_RS** όπου διαβάζουμε τον καταχωρητή rs από την Register File, έπειτα έχουμε την **BNEM_LD_RD** όπου διαβάζουμε τον καταχωρητή rd και κάνουμε σύγκριση τις διευθύνσεις της μνήμης που δείχουν οι καταχωρητές **rs** και **rd**. Σε αυτό μας βοηθάει το module **QUADREG** το οποίο σε αυτή την περίπτωση λειτουργεί ως ένας απλός καταχωρητής που κρατάει τα δεδομένα της εξόδου της μνήμης. Έτσι, σε **2 κύκλους** μπορούμε να συγκρίνουμε τα δεδομένα 2 διαφορετικών διευθύνσεών της. Τέλος, υπάρχει η κατάσταση **BNEM_PC** στη οποία επιλέγουμε τον **PC** δηλαδή αν θα διακλαδωθεί το πρόγραμμα ή όχι.

-byte_pack_mem(I-TYPE): Ξεκινάει από την κατάσταση **PACK** όταν έρθει το αντίστοιχο Instruction("**111100**"). Συνεχίζει με τις καταστάσεις **PACK_A**, **PACK_B**, **PACK_C** και **PACK_D** κατά τις οποίες στέλνουμε σε **4 διαδοχικούς κύκλους** τα δεδομένα 4 διαδοχικών διευθύνσεων(το **τελευταίο byte**) της μνήμης στο module **QUADREG** το οποίο κάνει pack τα δεδομένα στον καταχωρητή **rd** τον οποίο και γράφουμε στην Register File στην τελική κατάσταση η οποία είναι η **PACK_WR**.

-byte_unpack_mem(I-TYPE): Όπως και οι υπόλοιπες εντολές, όταν έρθει το κατάλληλο Instruction("**111110**") φτάνει στην κατάσταση **UNPACK**. Έπειτα, έχουμε τις καταστάσεις **UNPACK_A**, **UNPACK_B**, **UNPACK_C** και **UNPACK_D** κατά τις οποίες στέλνουμε με τη σειρά σε 4 διαδοχικούς κύκλους τα δεδομένα του καταχωρητή **rd, byte-byte** στο module **UNPACKER** το οποίο τα αποθηκεύει σε 4 διαδοχικές θέσεις μνήμης στο οποίο γίνεται και το κατάλληλο **sign extension.**

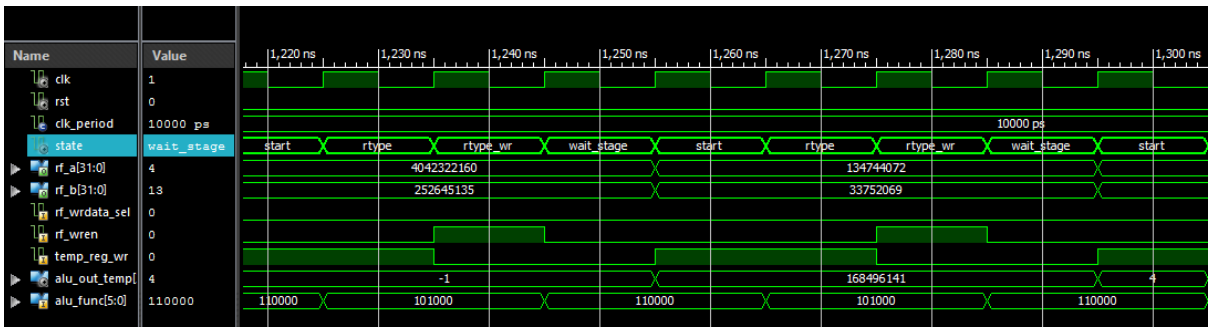
Κυμματομορφές:

CMOV :



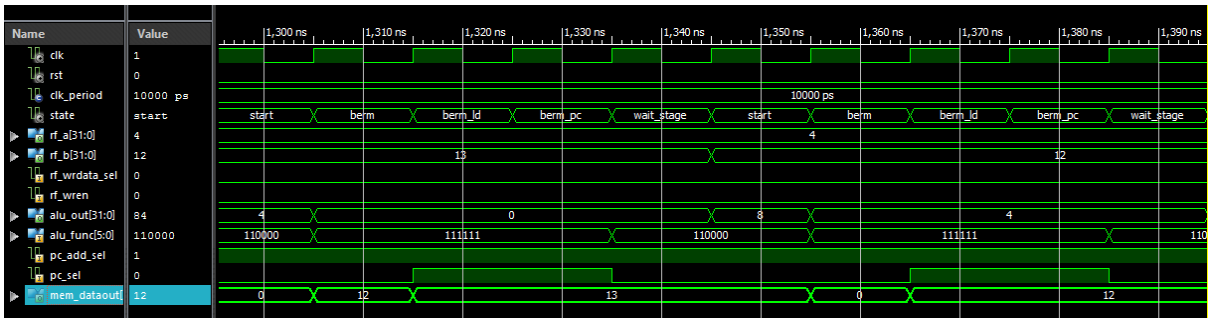
Στην παραπάνω κυμματομορφή βλέπουμε 3 διαδοχικές CMOV. Αν προσέξουμε τα σήματα rf_a και alu_out_temp θα προσέξουμε οτι η ALU εξάγει τον RF[rs] ατόφιο. Επίσης μπορούμε να δούμε οτι το σήμα rf_wren ενεργοποιείται στην πρώτη και στην τρίτη περίπτωση, όχι όμως στην δεύτερη όπου εχουμε rf_b = 0.

add_MMX_byte :



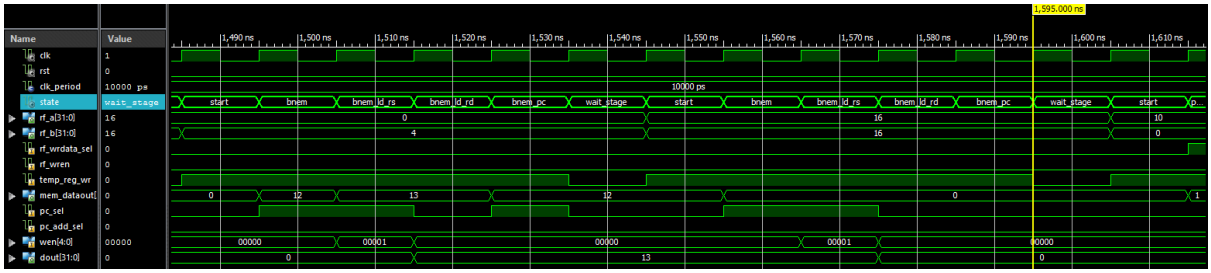
Η add_MMX_byte θεωρείται rtype εντολή με μόνη διαφορά το function της ALU όπου στην προκειμένη περίπτωση είναι “101000”. Ετσι η ALU ξερει να κανει κατάλληλη πράξη. Όλα τα άλλα σήματα είναι ίδια με το προηγούμενο εργαστήριο.

branch_equal_reg_mem :



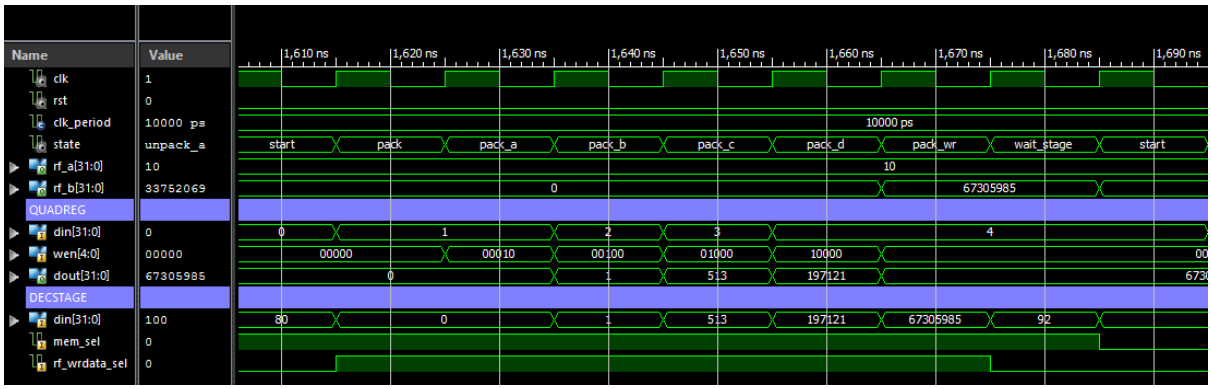
Εδω βλέπουμε μια BERM. Αρχικά μπορούμε να δούμε το σήμα pc_add_sel = 1 το οποίο και λέει στην IFSTAGE πως αν πρόκειται να γίνει branch τότε προσθέτουμε στο pc το 4 + RF[rs]. Ακόμη βλέπουμε οτι αν τα σήματα mem_dataout και rf_b ισούται το σήμα pc_sel γινεται 1 άρα έχουμε branch.

branch_not_equal_mem:



Σε αυτήν την περίπτωση χρησιμοποιούμε το module “QUADREG” (wen = 00001) ωστε να κρατήσουμε τα δεδομένα της μνήμης που βρίσκονται στην θεση RF[rs], μετά απο έναν κύκλο τα συγκρίνουμε με τα δεδομένα της μνήμης που βρίσκονται στην θέση RF[rd] και έτσι επιλέγουμε αν θα γίνει branch με το σήμα pc_sel.

byte_pack_mem:



Το module “QUADREG” χρησιμοποιείται ξανά αλλα αυτήν την φορά χρησιμοποιώντας διαφορετικές τιμές στο σήμα wen. Αυτές οι τιμές λένε στο module να κρατήσει το πρώτο byte της εισόδου του και να το αποθηκεύσει (και εξάγει) στο αντίστοιχο byte της εξόδου του . Για παράδειγμα wen=00100 σημαίνει ότι το Module θα πάρει το πρώτο byte της εισόδου του και θα το εξάγει ώς το δεύτερο byte της εξόδου του, χωρίς να πειράξει τα υπόλοιπα bytes. Στην περίπτωση εισόδου wen = 00001 ο quadreg λειτουργεί ώς ένας απλός 32bit register. Επίσης βλέπουμε το σήμα mem_sel = 1 το οποίο θέτει ως είσοδο του RF τα δεδομένα του quadreg και όχι του memstage.

byte_unpack_mem:

Παραπάνω βλέπουμε την εντολή unpack. Για αυτήν την εντολή χρησιμοποιείται το Module “unpacker” το οποίο χρησιμοποιώντας το σήμα control παίρνει το αντίστοιχο byte απο την είσοδό του και το βγάζει στην έξοδο sign extended. Βλέπουμε ακόμη οτι το mem_wren είναι στον άσσο για 4 διαδοχικούς κύκλους κάνοντας 4 διαδοχικές εγγραφές. Για την διαδοχική αλλαγή του memory address χρησιμοποιήθηκε ως address το σήμα immediate της decstage στο οποίο και προστίθεται το σήμα immed_offset. Με ομοιο τρόπο αλλάζουν και οι διαδοχικές διευθύνσεις της μνήμης στην εντολή byte_pack_mem:

Συμπεράσματα:

Στα πλαίσια της 4ης εργαστηριακής άσκησης ασχοληθήκαμε με τον τρόπο με τον οποίο υλοποιούμε έναν επεξεργαστή πολλαπλών κύκλων που εκτελεί σύνθετες εντολές. Αυτό το πετύχαμε επεκτείνοντας το προηγούμενο datapath και προσθέτοντας επιπλέον καταστάσεις στην FSM που αποτελεί το Control του επεξεργαστή.