Εργασία #2:
Σχεδίαση επεξεργαστή πολλαπλών κύκλων και μετατροπή του σε pipeline

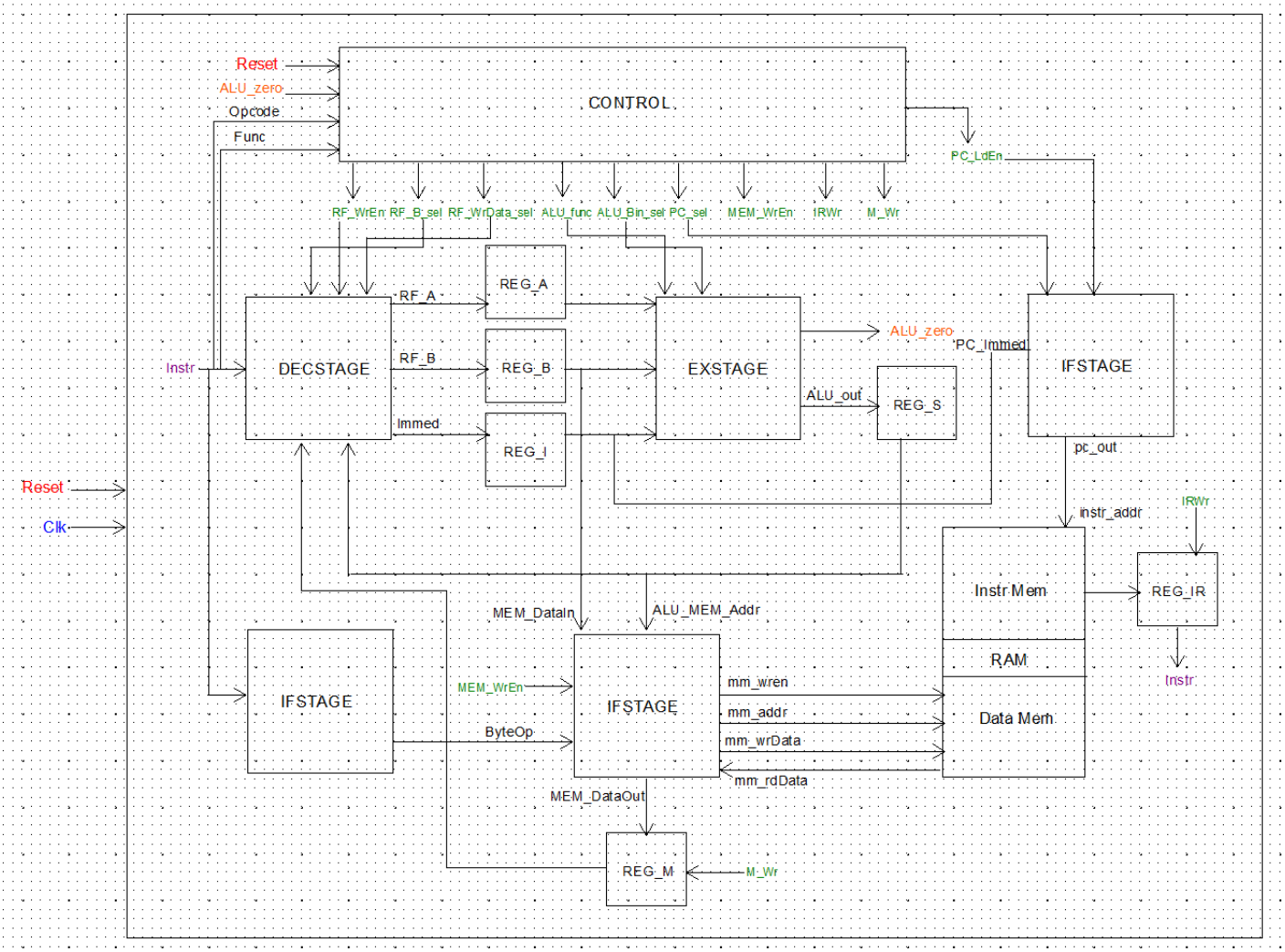
Μανουδάκη Χρυσηίς 2019030201

Φάση 4η: Σχεδίαση επεξεργαστή πολλών κύκλων

Στην τέταρτη φάση της άσκησης, ασχοληθήκαμε με την μετατροπή του επεξεργαστή ενός κύκλου σε επεξεργαστή πολλαπλών κύκλων. Αυτό επιτεύχθηκε με την χρήση ενός μικρού αριθμού καταχωρητών ώστε να αποθηκεύονται μερικά σημαντικά σήματα που θα χρειαζόμαστε στον επόμενο κύκλο. Η βασική διαφορά με την προηγούμενη εργαστηριακή άσκηση, είναι ότι με αυτή την αρχιτεκτονική κάθε εντολή καταλαμβάνει συγκεκριμένο αριθμό κύκλων ρολογιού, π.χ η Load εντολή θέλει 5 κύκλους ενώ η R-Type εντολές θέλουν 4 κύκλους. Με αυτόν τον τρόπο, δημιουργούμε έναν επεξεργαστή που εκτελεί πιο γρήγορα τις εντολές (μικρότερο CPI) από τον Single-Cycle επεξεργαστή.

Προεργασία

Παρακάτω απεικονίζεται το διάγραμμα του επεξεργαστή πολλών κύκλων:



Περιγραφή

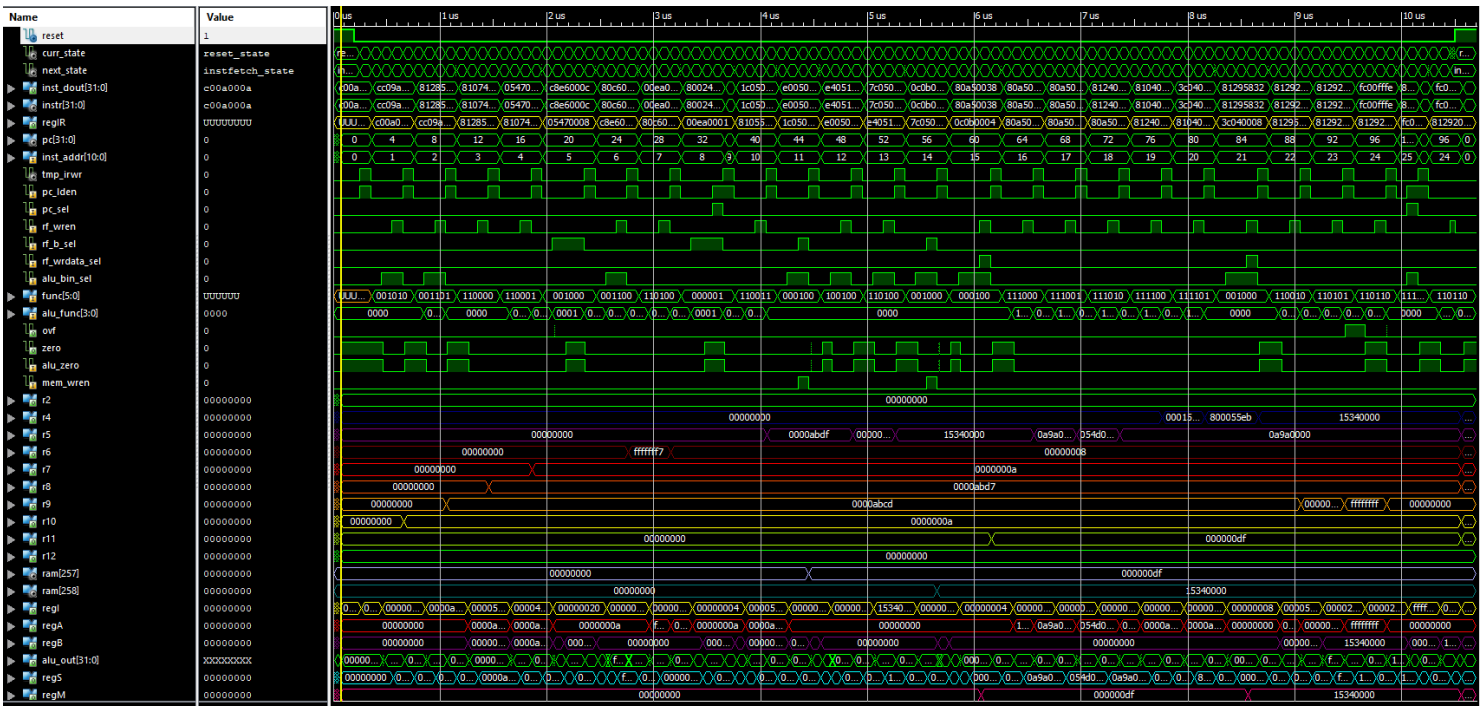
Η υλοποίηση αυτής της αρχιτεκτονικής δεν παρουσιάζει σημαντικές διαφορές από αυτή του επεξεργαστή ενός κύκλου. Χωρίζεται επίσης σε τρία κομμάτια: το Datapath, το Control και το top level module που περιέχει τα δύο προηγούμενα, Processor_Mc. Η σύνδεση με την μνήμη γίνεται στο Testbench του Processor_Mc. Ειδοποιός διαφορά, είναι η προσθήκη 6 καταχωρητών 32 bits οι οποίοι αποθηκεύουν τα απαραίτητα δεδομένα για να προχωρήσει μία εντολή στον επόμενο κύκλο. Οι καταχωρητές REG_A, REG_B και REG_I αποθηκεύουν τις εξόδους του DECSTAGE, RF_A, RF_B και Immed αντίστοιχα. Ο καταχωρητής REG_S αποθηκεύει την εξόδο ALU_out από το EXSTAGE, ο καταχωρητής REG_IR αποθηκεύει την εντολή Instr που παίρνεται από την μνήμη και τέλος στον καταχωρητή REG_M, αποθηκεύονται τα δεδομένα που διαβάστηκαν από την μνήμη. Οι δύο τελευταίοι καταχωρητές, έχουν τα σήματα ενεργοποίησης IRWr και M_Wr, τα οποία παράγονται από το CONTROL και ενεργοποιούνται όταν θέλουμε να πάρουμε νέα εντολή και όταν θέλουμε να διαβάσουμε από την μνήμη αντίστοιχα. Μία ακόμα αλλαγή που έγινε στην αρχική αρχιτεκτονική της εργασίας 1, είναι η αλλαγή στις εισόδους του Adder στο IFSTAGE που χρησιμοποιείται για τις Branch εντολές. Οι νέες εισόδοι είναι το PC_Immed και το PC_out χωρίς την αύξηση κατά τέσσερα. Αυτό έγινε διότι παρουσιαζότανε δυσκολία κατά τις branch εντολές όσον αφορά τον συγχρονισμό της διεύθυνσης και της εντολής. Στο CONTROL υλοποιήθηκε μία μηχανή πεπερασμένων καταστάσεων (FSM) και πιο συγκεκριμένα το μοντέλο Moore καθώς οι έξοδοι υπολογίζονται συναρτήσει μόνο της παρούσας κατάστασης. Με αυτό τον τρόπο, κατορθώθηκε κάθε εντολή ανάλογα με τον τύπο που είναι π.χ. R-Type, να εκτελείται στους αντίστοιχους κύκλους ρολογιού που χρειάζεται. Έτσι, όπως παρουσιάζεται παραπάνω, το CONTROL λαμβάνει μία είσοδο (OPCODE) και αντίστοιχα ακολουθεί την πορεία κάθε γενικού τύπου εντολής. Με άλλα λόγια, το CONTROL ξεκινάει με ένα Instruction State στη συνέχεια οδηγείται στο Decode State και από εκεί ανάλογα με το OPCODE οδηγείται στα R-Type State,Branch State και Load-Store State,όπου σε κάθε ένα από αυτά ενεργοποιούνται τα απαραίτητα σήματα ελέγχου(Flags). Στο στάδιο της κοινής μνήμης (RAM) δεν υπήρξε καμία αλλαγή και ήταν ίδια με την προηγούμενη εργαστηριακή άσκηση, τόσο στο port mapping όσο και στο κομμάτι του κώδικα. Τέλος, ο τελικός επεξεργαστής είχε παρόμοιο port mapping με την προηγούμενη εργαστηριακή άσκηση ,απλά υπήρχαν ελάχιστες διαφορές σε ορισμένα σήματα, όσον αφορά στην ονομασία τους.

Για το testbench του PROCESSOR_MC δημιουργήθηκε ένα πρόγραμμα αναφοράς το οποίο περιλαμβάνει όλες τις εντολές του ISA τουλάχιστον από μία φορά.

00: addi r10, r0, 10	//r10 = 10
04: ori r9, r0, 0xABCD	//r9 = 0x0000ABCD
08: add r8, r9, r10	//r8 = 0x0000ABD7
12: sub r7, r8, r9	//r7 = 10
16: bne r7, r10, 8	//r7=r10= 10 άρα αποτυχημένη διακλάδωση
20: nandi r6, r6	// r6 = 0xfffffff7
24: not r6, r7, 12	//r6 = 8
28: beq r10, r7, 12	//r10=r7=10 θα εκτελεστεί, άρα θα διακλαδωθεί στην εντολή 36
32: add r2, r0, r9	// δεν θα εκτελεστεί
36: or r5, r8, r10	// r5 = 0x0000ABDF
40: sb r5, 4(r0)	// γράφει στην διεύθυνση 0x404 την τιμή 0x000000DF

44: li r5, 0x64	//r5 = 0x00000064
48: lui r5, 0x1534	// r5 = 0x15340000
52: sw r5, 8(r0)	// γράφει στην διεύθυνση 0x408 την τιμή 0x15340000
56: lb r11, 4(r0)	// r11 = 0x000000DF
60: sra r5, r5	// r5 = 0x0A9A0000
64: srl r5, r5	// r5 = 0x054D0000
68: sll r5, r5	// r5 = 0x0A9A0000
72: rol r4, r9	// r4 = 0x0001579A
76: ror r4, r8	// r4 = 0x800055EB
80: lw r4, 8(r0)	// r4 = 0x15340000
84: and r9, r9, r11	// r9 = 0x000000CD
88: nand r9, r9, r4	// r9 = 0xffffffff
92: nor r9, r9, r4	// r9 = 0
96: b -2	// εκτελεί την εντολή 92. Infinite loop!
100: srl r5, r5	// δενθα εκτελεστεί

Η κυματομορφή που απεικονίζονται τα παραπάνω:

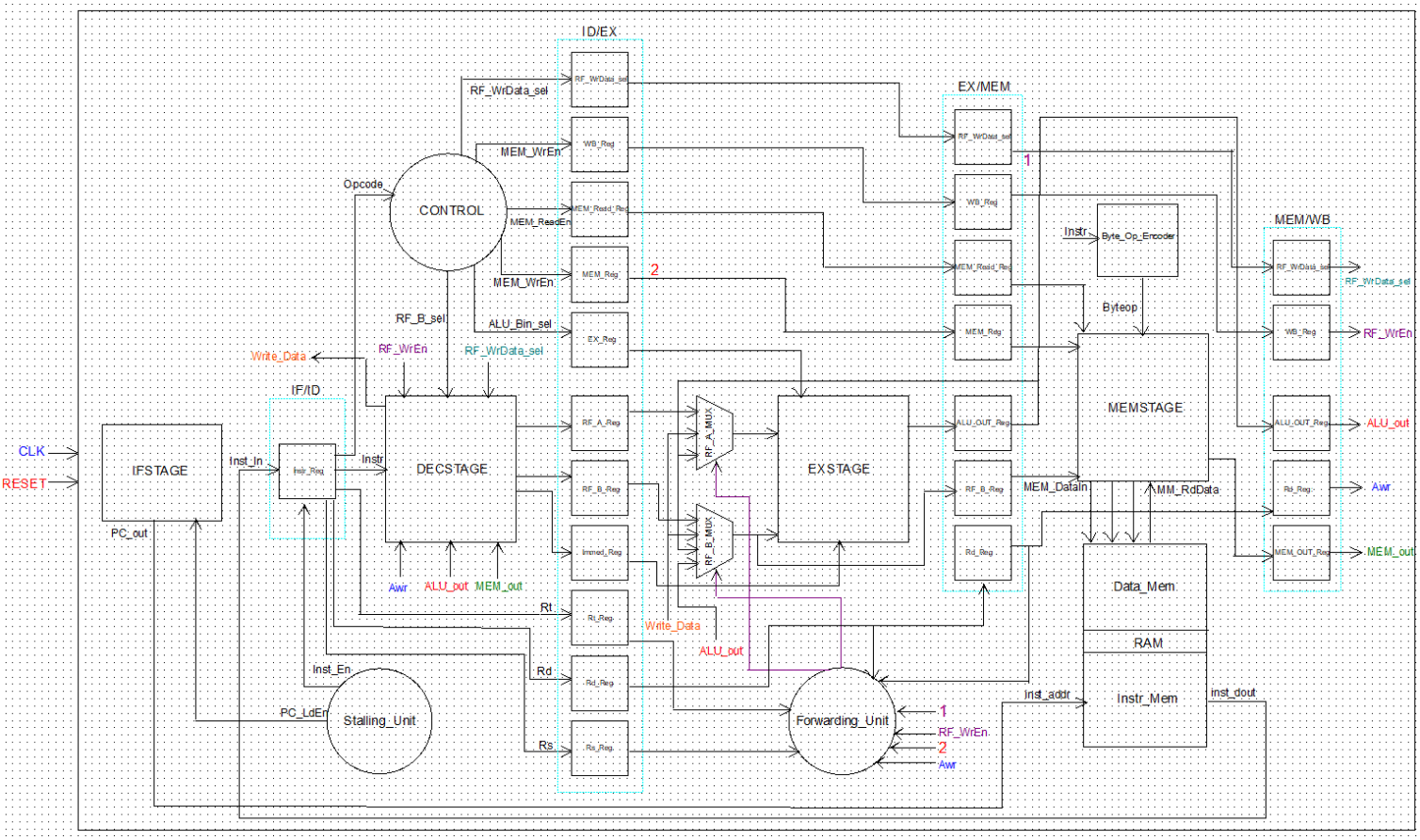


Φάση 5η: Σχεδίαση επεξεργαστή pipeline

Στο 2ο μέρος της άσκησης ασχοληθήκαμε με τη δημιουργία ενός επεξεργαστή με αρχιτεκτονική Pipeline,ο οποίος θα αποτελείται από μεγάλο πλήθος καταχωρητών ώστε να διαχειρίζονται πιο εύκολα οι τιμές των σημάτων και να διατηρούνται για ένα κύκλο ρολογιού. Με αυτό τον τρόπο επιτυγχάνεται να μην μένει κανένα στάδιο του επεξεργαστή αχρησιμοποίητο καθώς εκτελούνται συνέχεια εντολές με διαφορά μόλις ενός κύκλου. Αναμενόμενο λοιπόν ήταν η υλοποίηση της αρχιτεκτονικής να είναι πιο δύσκολη και χρονοβόρα αλλά στο τέλος φαίνεται ότι ο χρόνος τρεξίματός ενός προγράμματος αναφοράς σε αυτό τον επεξεργαστή θα τρέξει έως και 5 φορές πιο γρήγορα από τους 2 προηγούμενους που υλοποιήθηκαν. Στη συγκεκριμένη εργαστηριακή άσκηση ζητούσαν μόνο τέσσερις εντολές (Li,Lw,Sw,Add), το οποίο απλοποίησε και διευκόλυνε πολύ το έργο μας.

Προεργασία

Παρακάτω απεικονίζεται το διάγραμμα του pipeline επεξεργαστή:



Περιγραφή

Η υλοποίηση αυτής της αρχιτεκτονικής είναι πιο απαιτητική από αυτή του επεξεργαστή ενός κύκλου και μοιάζει περισσότερο με αυτήν των πολλών κύκλων. Συγκεκριμένα, προστέθηκαν μετά από κάθε stage, βαθμίδες οι οποίες μέσα σε καταχωρητές αποθηκεύουν για έναν κύκλο σχεδόν όλα τα σήματα που προέρχονται από το προηγούμενο stage. Πέρα από αυτό, οι υπόλοιπες αλλαγές είναι μικροαλλαγές σε υποκυκλώματα και τα νέα δύο κυκλώματα Forwarding_Unit και Stalling_Unit τα οποία αντιμετωπίζουν Data Hazards. Στο DECSTAGE η διαφορά από το αρχικό κύκλωμα είναι η προσθήκη ενός σήματος εξόδου Write_Data το οποίο επιστρέφει τα δεδομένα (ALU_out ή MEM_out) που θα γραφτούν στον καταχωρητή. Επίσης, πλέον ως διεύθυνση καταχωρητή για εγγραφή δίνεται το Rd από το στάδιο MEM/WB. Στο στάδιο του EXSTAGE, προστέθηκαν δύο πολυπλέκτες και το Forward_Unit. Το Forward_Unit έχει ως εισόδους τα ID/EX_RT, ID/EX_RS, EX/MEM_RD, EX/MEM_WB_reg, MEM/WB_WB_reg,

ID/EX_MEM_reg και MEM/WB_Rd_Reg και ως εξόδους παράγει δύο σήματα ελέγχου, τα οποία εισέρχονται στους αντίστοιχους πολυπλέκτες. Σε αυτό το υποκύκλωμα οι δύο πολυπλέκτες που προαναφέρθηκαν έχουν ως σήματα εισόδου τις εξόδους του REGISTER FILE(RF_A,RF_B) , την έξοδο της ALU από το EX/MEM στάδιο και από το MEM/WB στάδιο και τέλος, την έξοδο Write_Data από το DECSTAGE. Η δουλειά του συγκεκριμένου υποκυκλώματος είναι να λειτουργεί στην ειδική περίπτωση που ο τελικός πολυπλέκτης δεν έχει προλάβει να δώσει την τιμή του στο δεύτερο υποκύκλωμα (DEC), με αυτόν τον τρόπο και όταν ισχύσει η ειδική περίπτωση, τα σήματα εξόδου των δύο (2) πολυπλεκτών θα προωθούν τις τιμές τους στις παρακάτω βαθμίδες του Pipeline 3η , 4η (EX/MEM, MEM/WB), έτσι ώστε να αποφεύγονται data hazards. Το δεύτερο επιπλέον υποκύκλωμα που προστέθηκε στο DATAPATH αυτό είναι το Stalling_Unit. Αυτό το κύκλωμα έχει εισόδους τα σήματα ID/EX_Mem_Read_reg, ID/EX_Rd, IF/ID_Rs, IF/ID_Rt και εξόδους το σήμα Pc_LdEn που ενεργοποιεί τον καταχωρητή PC, και το Instr_Reg_En που ενεργοποιεί τον καταχωρητή της βαθμίδας IF/ID. Ο ρόλος αυτού του υποκυκλώματος, είναι να προλαμβάνει, σε περίπτωση Load Delay Slot, να βάζει τα λεγόμενα BUBBLES, δηλαδή να απενεργοποιεί τον PC COUNTER για ένα κύκλο ρολογιού. Στο στάδιο του CONTROL τα πράγματα ήταν πολύ πιο εύκολα, λόγω των ελάχιστων εντολών που είχαμε από την εκφώνηση. Με άλλα λόγια, ο βασικός κορμός του CONTROL παρέμεινε ο ίδιος με αυτό της 1ης εργαστηριακής άσκησης (SINGLE- CYCLE PROCESSOR), με μόνη διαφορά το γεγονός ότι είχαμε να υλοποιήσουμε μόνο 4 εντολές. Στο στάδιο της κοινής μνήμης (RAM) δεν υπήρξε καμία αλλαγή και ήταν ίδια με την προηγούμενη εργαστηριακή άσκηση, τόσο στο port mapping όσο και στο κομμάτι του κώδικα. Τέλος, ο τελικός επεξεργαστής είχε παρόμοιο port mapping με την προηγούμενη εργαστηριακή άσκηση ,απλά υπήρχαν ελάχιστες διαφορές σε ορισμένα σήματα, όσον αφορά στην ονομασία τους.

Για το testbench του PROCESSOR_PIPELINE δημιουργήθηκε ένα πρόγραμμα αναφοράς το οποίο περιλαμβάνει μόνο τις εντολές li, lw, sw, add πολλές φορές ώστε να φανεί η λειτουργία του pipeline και η αντιμετώπιση των data hazards.

00: li r1, 8	//r1 = 8
04: li r2, 10	//r2 = 10
08: add r3, r1, r2	//r3 = 18 --forwarding
12: add r4, r3, r2	//r4 = 28 --forwarding
16: add r5, r3, r1	//r5= 26 -- forwarding
20: sw r2, 4(r0)	// γράφει στην διεύθυνση 0x404 την τιμή 10
24: sw r5, 8(r0)	// γράφει στην διεύθυνση 0x408 την τιμή 26 --forwarding
28: lw r6, 4(r0)	// r6 = 10
32: add r1, r6, r4	// r1 = 38 -- stalling
36: sw r1, 0(r0)	// γράφει στην διεύθυνση 0x400 την τιμή 38 --forwarding
40: lw r3, 0(r0)	// r3 = 38
44: add r5, r3, r2	// r5 = 48 --stalling

Η κυματομορφή που απεικονίζονται τα παραπάνω: