Οργάνωση Υπολογιστών

Εργασία 1

Παπαδόπουλος Αλέξανδρος 2014030071

Σε αυτή την εργασία στόχος μας είναι η σχεδίαση ενός επεξεργαστή single cycle, που σημαίνει ότι όλες οι εντολές εκτελούνται σε έναν κύκλο ρολογιού. Για να καταλήξουμε στο αποτέλεσμα χρειάστηκε επιπλέον: εξοικείωση μας με τη VHDL γλώσσα, επανάληψη ψηφιακών κυκλωμάτων και Assembly γλώσσας προγραμματισμού και τέλος, η κατανόηση της ιεραρχικής σχεδίασης η οποία βοηθάει στο να γίνεται ευκολότερα και γρηγορότερα η αποσφαλμάτωση η οποία παίζει τεράστιο ρόλο στον έλεγχο της σωστής λειτουργίας όλων των κυκλωμάτων που απαρτίζουν τον τελικό μας επεξεργαστή.

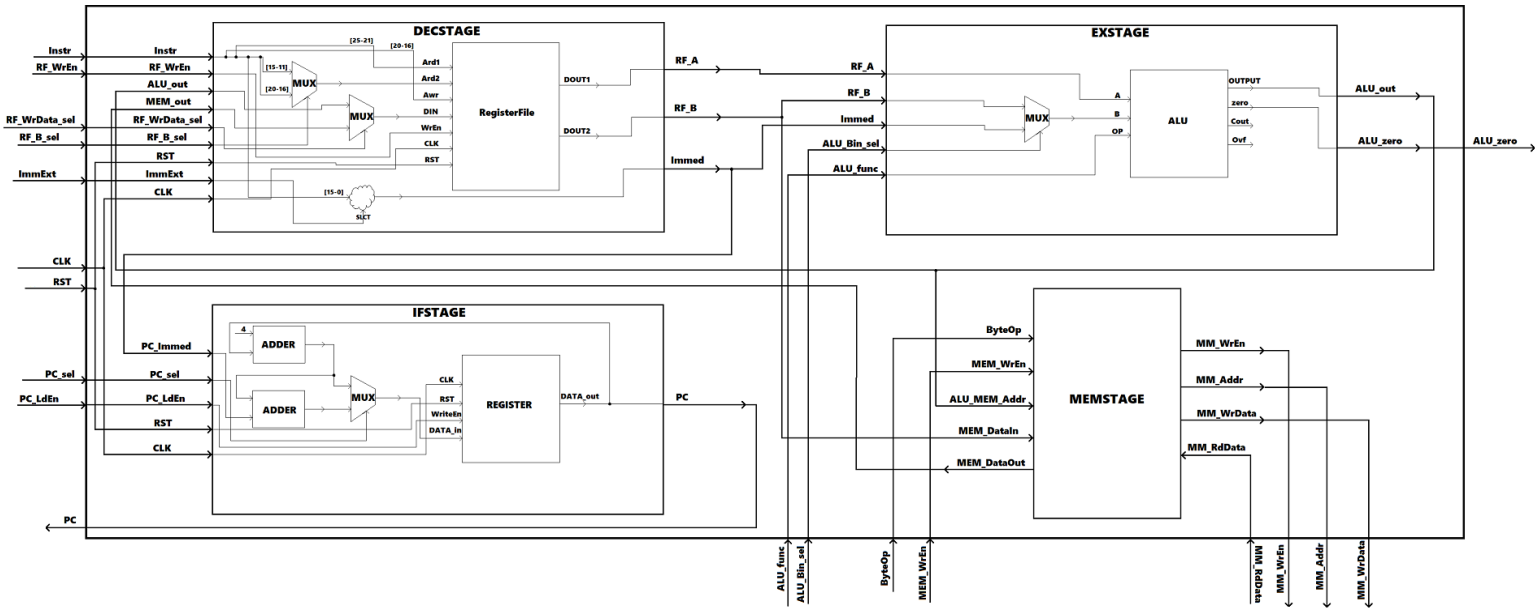
Στην πρώτη φάση σχεδίασα αρχικά τη μονάδα αριθμητικών και λογικών πράξεων ALU με Behavioral architecture.Η ALU δέχεται δύο 32 Bit εισόδους A και B, ανάλογα με την είσοδο Op πράττει και την ανάλογη πράξη μεταξύ A και B. Έλεγξα τη λειτουργία της μέσω ενός εξαντλητικού Test Bench. Κατασκεύασα επίσης το αρχείο καταχωρητών η αλλιώς Register File. Σχεδίασα έναν Register 32 Bit, έναν Decoder 5 to 32 και έναν Multiplexer 32 to 1,στη συνέχεια με Structural architecture, δημιούργησα και ένωσα 32 Registers με τα κατάλληλα σήματα, με τη χρήση του “for generate”. Επιπλέον, σύνδεσα τον Decoder, τους 2 Registers εξόδου και τους Multiplexers με κατάλληλα σήματα. Ο Decoder είναι υπεύθυνος για την επιλογή του Register που θα εγγραφούν τα δεδομένα, ενώ οι δύο πολυπλέκτες είναι υπεύθυνοι για το ποίος Register θα βγει στην έξοδο. Τέλος, για τον Multiplexer κατασκεύασα δικό μου package και όρισα ένα νέο type. Δημιούργησα έτσι έναν πίνακα που περιέχει vectors.

Στη δεύτερη φάση ξεκίνησα με τον έλεγχο της RAM που μας δόθηκε έτοιμη από την εκφώνηση της εργασίας. Με εξαντλητικό Test Bench έγινε η εξακρίβωση της σωστής λειτουργίας της. Συνέχισα με την κατασκευή της βαθμίδας ανάκλησης εντολών IFSTAGE. Δημιούργησα έναν ADDER 32 Bit και έναν Multiplexer των 32 Bit επίσης 2 to 1. Με Structural architecture και με port map συνδέθηκαν οι 2 ADDER, ένας για να μετρά διαδοχικά προσθέτοντας απλά 4 στον PC (Program Counter) και ένας ο οποίος είναι υπεύθυνος για branch και jump εντολές και προσθέτει 4 και Immed στον PC. Επίσης, συνδέθηκε με παρόμοιο τρόπο και ο Multiplexer με τον οποίο ελέγχουμε ποιος ADDER θα μετρήσει και θα αποθηκευτεί στον Register. Για τη βαθμίδα αποκωδικοποίησης εντολών DECSTAGE αρχικά χρειάστηκε να σχεδιάσω έναν 5 Bit Multiplexer 2 to 1 ο οποίος είναι υπεύθυνος για την επιλογή είτε του rd είτε του rt, τα (20-16) και (15-11) bits δηλαδή της Instruction. Έγινε χρήση επίσης ενός 32 Bit Multiplexer 2 to 1, υπεύθυνος για την επιλογή των δεδομένων εγγραφής. Στη συνέχεια σχεδίασα τη μονάδα Immed_extend η οποία ανάλογα την είσοδο Select κάνει: Sign Extention, Zero fill τα 16 Most Significant Bits και shift left 2. Με port map συνδέθηκαν όλα τα επιμέρους στοιχεία μαζί και με τον Register File και στη συνέχεια έγινε εξαντλητικός έλεγχος της λειτουργίας του DECSTAGE μέσω εξαντλητικού Test Bench. Για τη βαθμίδα εκτέλεσης εντολών EXSTAGE χρησιμοποίησα την ALU που σχεδίασα στην πρώτη φάση της εργασίας και έναν Multiplexer 32 bit 2 to 1 που είναι υπεύθυνος για την επιλογή του RF_B σε R-TYPE εντολές και του Immed σε I-TYPE εντολές, η επιλογή γίνεται με το σήμα

ALU_Bin_sel. Στην ALU περνάμε μέσω του ALU_func τον κωδικό πράξης και παίρνουμε το αποτέλεσμα στην έξοδο. Όλα συνδέθηκαν μεταξύ τους με Structural architecture με port map. Τη βαθμίδα πρόσβασης στη μνήμη MEMSTAGE η οποία είναι υπεύθυνη για την ανάγνωση και την εγγραφή δεδομένων από και προς τη RAM, τη σχεδίασα με Behavioral τρόπο. Στο ALU_MEM_Addr προσθέτουμε το δεκαεξαδικό 0x400 και το περνάμε στην έξοδο MM_Addr με την οποία δίνουμε στη RAM τη διεύθυνση των δεδομένων. Το DATA segment βρίσκεται 256 θέσεις μετά το Text segment για αυτό και γίνεται η πρόσθεση 0x400. Το Mem_WrEn είναι υπεύθυνο για το αν θα γίνει εγγραφή δεδομένων στη μνήμη και συνδέεται με το MM_WrEn. Το ByteOp υποδεικνύει αν θα γίνει εγγραφή - ανάγνωση word ή byte οπότε είναι ένα σήμα ελέγχου και τέλος το MEM_DataIn δέχεται τα δεδομένα που θα γραφτούν μέσω του MM_WrData στη RAM και το MEM_DataOut δίνει δεδομένα στο DECSTAGE από τη RAM μέσω του MM_RdData. Έτσι δημιούργησα το top level MEMSTAGE.

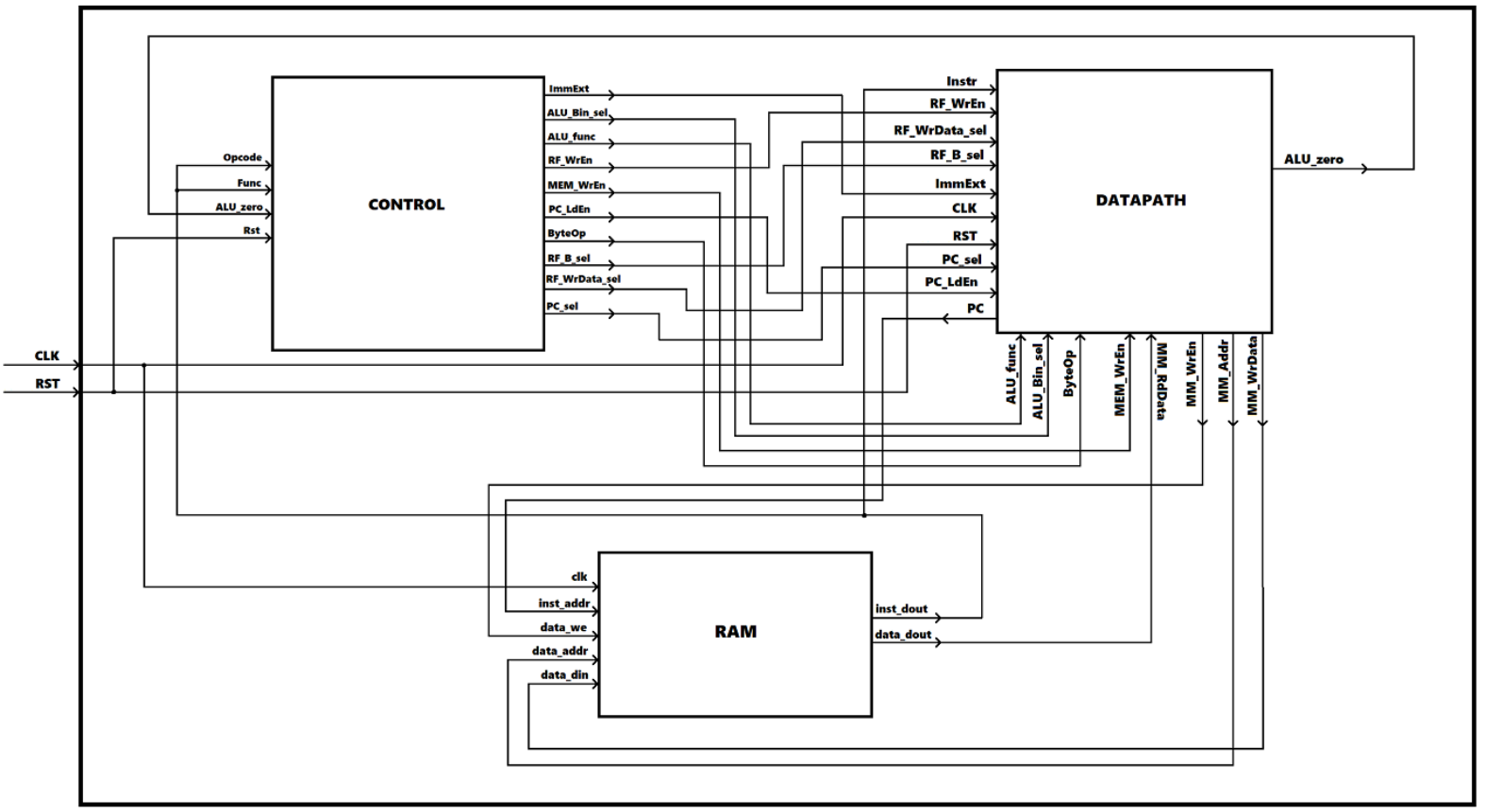
Στην τρίτη φάση σύνδεσα τα: IFSTAGE, DECSTAGE, EXSTAGE και MEMSTAGE σε ένα top level module, το DATAPATH, με Structural architecture όπως φαίνεται στην παρακάτω εικόνα.

DATAPATH



Για την επαλήθευση της σωστής λειτουργίας του DATAPATH, δημιούργησα ένα Test Bench στο οποίο προσέθεσα ένα instance της RAM με component και με port map σύνδεσα RAM και DATAPATH καταλλήλως. Με χρήση του προγράμματος αναφοράς 1, έγινε η επαλήθευση της σωστής λειτουργίας του DATAPATH. Σχεδίασα το CONTROL με Behavioral architecture με process και cases. Το CONTROL είναι υπεύθυνο για τα σήματα ελέγχου του DATAPATH τα οποία παράγει ανάλογα με την είσοδο Opcode. Το Opcode είναι τα πρώτα 6 Bits του Instruction που διαβάζεται από τη RAM. Για την επαλήθευση της σωστής λειτουργίας του CONTROL έφτιαξα επίσης ένα Test Bench με όλες τις διαφορετικές τιμές που θα μπορούσε να πάρει το Opcode. Τέλος, σε ένα Top Level Module, το PROC_SC, έγινε η σύνδεση του CONTROL του DATAPATH και της RAM όπως φαίνεται στην παρακάτω εικόνα:

PROC_SC

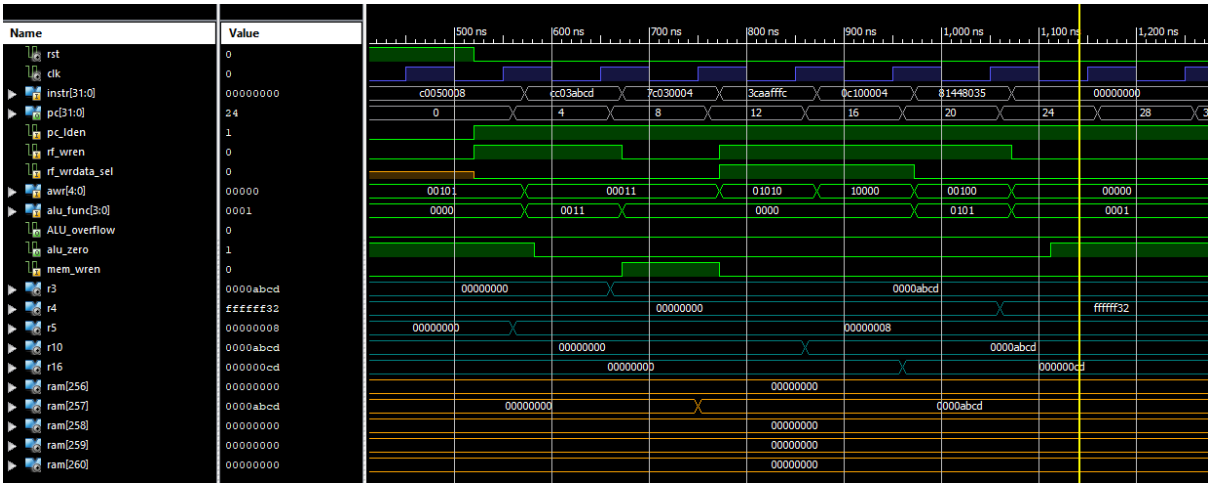


Για την επαλήθευση της σωστής λειτουργίας του PROC_SC, δημιούργησα ένα Test Bench και με τη χρήση δύο διαφορετικών ROM.data αρχείων που περιείχαν τα 2 προγράμματα αναφοράς που δόθηκαν στην εκφώνηση της εργασίας, έκανα την επαλήθευση παρακολουθώντας τα σωστά σήματα που ζητήθηκαν από την εκφώνηση στα WAVEFORMS.

Αποτελέσματα Προγραμμάτων Αναφοράς

Πρόγραμμα αναφοράς 1:

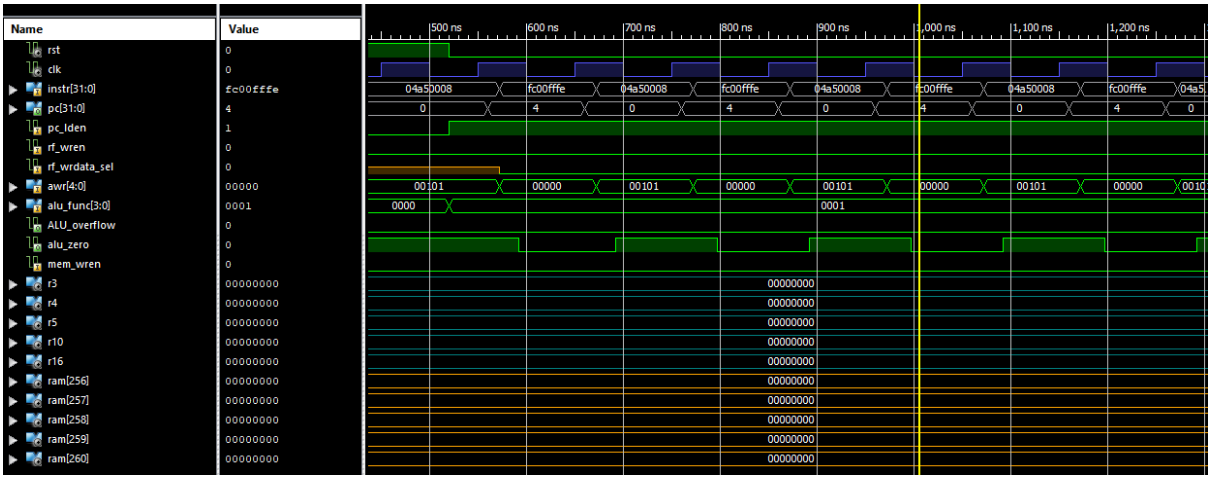
```
00: addi r5, r0, 8
04: ori  r3, r0, 0xABCD
08: sw   r3, 4(r0)        // γράφει στην διεύθυνση 0x4 => 0x404 την τιμή 0x0000ABCD
0C: lw  r10, -4(r5)       // διαβάζει από την διεύθυνση 0x4 => 0x404 την τιμή 0x0000ABCD
10: lb  r16, 4(r0)        // διαβάζει byte από την διεύθυνση 0x4 => 0x404 την τιμή 0x000000CD
14: nand r4, r10, r16
```



Ενεργοποίησα το Rst για 5 κύκλους ρολογιού και κατά την απενεργοποίηση του, το σήμα PC_LdEn ενεργοποιήθηκε, οπότε ενεργοποιήθηκε ο Program Counter. Για την πρώτη εντολή το ALU_func παίρνει την τιμή 0 έτσι ώστε να γίνει η πράξη ADD. Το ALU_Bin_sel ενεργοποιείται για να γίνει πράξη μεταξύ καταχωρητή και Immed εισόδου στην ALU εφόσον έχουμε I-TYPE εντολή. Το αποτέλεσμα της πράξης θα αποθηκευτεί στον r5, επομένως το Rf_WrData_sel = ’0’ για να γραφτούν τα δεδομένα που θα έρθουν από την ALU (EXSTAGE) και όχι από τη μνήμη (MEM). Το RF_WrEn είναι ενεργοποιημένο για να γραφτούν δεδομένα στο Register File. Το MEM_WrEn = ‘0’ διότι δε θέλουμε να αποθηκεύσουμε δεδομένα στη μνήμη. Το ImmExt παίρνει την τιμή 0 για να γίνει Sign Extention του Immed εφόσον η εντολή ADD δεν είναι unsigned. Τα δεδομένα της πράξης θα αποθηκευτούν στον καταχωρητή r5 και η επιλογή του γίνεται μέσω του Awr του Register File, που δέχεται τα (20-16) bits (rd) της Instruction και έτσι γίνεται η επιλογή του Register εγγραφής. Επιπλέον, το σήμα PC_sel παίρνει την τιμή 0 για την επιλογή του ADDER PC + 4 για να προχωρήσουμε στην επόμενη εντολή. Τα ίδια ισχύουν και για τη δεύτερη εντολή ori, με τη μόνη διαφορά ότι εδώ το ALU_func παίρνει την τιμή 3 για να γίνει η κατάλληλη πράξη (or). Στην τρίτη εντολή sw (store word) τα περιεχόμενα του καταχωρητή 3 θα αποθηκευτούν στη θέση 260 της μνήμης, αφού το data segment ξεκινά 256 θέσεις (λέξεις = 32 bits - 4 bytes) μετά το text segment. Σε αυτή την περίπτωση γίνεται πρόσθεση (αρα ALU_func = 0) του Immed και του καταχωρητή rs (25-21) bits που στην προκείμενη περίπτωση είναι ο r0. Αφού γίνει η πράξη από την ALU του EXSTAGE, με το ALU_Bin_sel να είναι και πάλι ενεργοποιημένο καθώς θέλουμε την Immed ποσότητα του DECSTAGE, μέσω της ALU_MEM_Addr υποδεικνύεται στη MEM σε ποια θέση θα γραφτούν τα δεδομένα που θα πάρουμε από το σήμα RF_B (άρα το RF_B_sel θα πάρει την τιμή 1 εφόσον θέλουμε να περάσουν τα [20-16] bits του rd) που καταλήγει στην είσοδο του MEMSTAGE τη MEM_DataIn. Το MEM_WrEn είναι ενεργοποιημένο καθώς θέλουμε να γίνει εγγραφή δεδομένων στη μνήμη. Επίσης, το ByteOp έχει την τιμή 0 καθώς γίνεται store μιας ολόκληρης λέξης δηλαδή 32 bits. Στην τέταρτη εντολή τα περιεχόμενα που υπάρχουν στη διεύθυνση [r5 - 4] της μνήμης θα αποθηκευτούν στον καταχωρητή r10. Μέσω λοιπόν του Awr γίνεται η επιλογή του r10 στο Register File της DECSTAGE. Η ALU πάλι θα υπολογίσει την πράξη πρόσθεσης του r5 και του -4, I-TYPE δηλαδή. Με την ALU_MEM_Addr είσοδο του MEMSTAGE θα γίνει πάλι διευθυνσιοδότηση της μνήμης και έτσι θα διαβαστούν από την τιμή 260 [r5 + (-4) = 8-4 = 4 -> 256 + 4 = 260] της διεύθυνσης της μνήμης τα δεδομένα. Στο DECSTAGE το RF_WrData_sel θα γίνει 1 εφόσον θέλουμε να γραφτούν τα δεδομένα από τη μνήμη στον Register File. Στην περίπτωση της πέμπτης εντολής load byte ισχύουν τα ίδια με τη διαφορά ότι τώρα είναι ενεργοποιημένο το ByteOp γιατί θέλουμε να φορτώσουμε 1 byte και όχι word (4 byte). Η τελευταία εντολή NAND είναι μια R-Type εντολή και άρα το ALU_Bin_sel θα είναι ίσο με 0 για να περάσει στην ALU ο δεύτερος καταχωρητής της εξόδου του DECSTAGE. Η func της ALU θα πάρει την τιμή της από το CONTROL (5-0 bits της instruction) για να γίνει η κατάλληλη πράξη. Όπως φαίνεται και στην παραπάνω εικόνα όλα τα σήματα που προκύπτουν από το Test Bench είναι αυτά που περίμενα και επομένως συμπεραίνω ότι είναι σωστή η λειτουργία του προγράμματος αναφοράς 1.

Πρόγραμμα Αναφοράς 2:

```
// *** Δεύτερο πρόγραμμα, μόνο διακλαδώσεις, εκτελεί δύο εντολές με αποτυχημένη διακλάδωση και ξανά
00: bne r5, r5, 8          // αποτυχημένη διακλάδωση
04: b -2                   // branch (PC=04 + 4 -2*4 = 00) infinite loop!
08: addi r1, r0, 1         // δεν θα εκτελεστεί
```



Ενεργοποίησα το Rst και εδώ για 5 κύκλους ρολογιού. Όταν επήλθαν οι 5 κύκλοι στην πρώτη εντολή bne που είναι μια εντολή διακλάδωσης, ελέγχεται αν οι δύο καταχωρητές έχουν διαφορετική τιμή, αυτό επιτυγχάνεται αφαιρώντας τους δύο καταχωρητές μεταξύ τους και αν είναι ίσοι το ALU_zero signal γίνεται 1 αφού το αποτέλεσμα της αφαίρεσης είναι 0. Αφού είναι ίσοι ο PC αυξάνεται μόνο κατά 4 λοιπόν. Στη δεύτερη εντολή έχουμε branch και το πρόγραμμα πάει μία θέση πίσω, σε αυτή την εντολή γίνεται επιλογή του Immed από το σήμα ALU_Bin_sel = ‘1’ και γίνεται Shift left με sign extention στην DECSTAGE, το PC_sel του IFSTAGE θα γίνει 1 ούτως ώστε να επιλεχθεί ο κατάλληλος ADDER και να γίνει η πράξη PC + 4 + Immed. Επομένως, όπως φαίνεται και επαληθεύεται από την παραπάνω εικόνα, το πρόγραμμα εκτελεί ατέρμονα αυτές τις 2 εντολές.

Στη συγκεκριμένη εργασία λοιπών κατανοήσαμε σε μεγαλύτερο βαθμό τη γλώσσα περιγραφής υλικού VHDL. Μάθαμε να χρησιμοποιούμε πακέτα που βοηθούν στην απλοποίηση του κώδικα μας, όπως για παράδειγμα είναι τα: IEEE.STD_LOGIC_UNSIGNED.ALL, IEEE.std_logic_arith.all, ieee.numeric_std.all. Καταλάβαμε ότι η ιεραρχική σχεδίαση βοηθάει στην αποσφαλμάτωση του κώδικα μας, με αποτέλεσμα να γίνεται γρηγορότερα, ευκολότερα και πιο σωστά. Κάναμε χρήση του after κάτι το οποίο βοήθησε να διαπιστώσουμε ότι σε πραγματικές συνθήκες υπάρχουν καθυστερήσεις και έτσι έγινε καλύτερη προσομοίωση ενός πραγματικού επεξεργαστή όπου για να βρούμε τη συχνότητα του Clock πρέπει να εντοπίσουμε το critical path της μεγαλύτερης εντολής σε διάρκεια χρόνου η οποία είναι η LOAD. Καταλάβαμε ότι ο έλεγχος όλων των module μέσα από εξαντλητικά Test Bench που καλύπτουν όλες τις περιπτώσεις είναι απαραίτητος και εξίσου σημαντικός ή και μέχρι ένα βαθμό σημαντικότερος από το κύκλωμα.