ΑΕΙ ΠΕΙΡΑΙΑ Τ.Τ.
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ Τ.Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση CPU σε VHDL

Αντώνιος – Παναγιώτης Παπαλεξάτος

Εισηγητής: Δρ Παύλος Κούρος

ΑΘΗΝΑ
2016

Αντώνιος-Παναγιώτης Παπαλεξάτος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση CPU σε VHDL

Αντώνιος-Παναγιώτης Παπαλεξάτος

Α.Μ. 39345

Εισηγητής: Δρ. Παύλος Κούρος

Εξεταστική Επιτροπή:

Ημερομηνία εξέτασης:

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα εργασία ολοκληρώθηκε μετά από αρκετές προσπάθειες και αφιέρωση χρόνου, με προσοχή στο γνωστικό αντικείμενο της μικροηλεκτρονικής. Τη συγκεκριμένη προσπάθεια ενίσχυσε με το καλύτερο τρόπο ο επιβλέπων καθηγητής μου κ. Παύλος Κούρος τον οποίο θα ήθελα να ευχαριστήσω για τις πολύτιμες γνώσεις του.

Επίσης θα ήθελα να ευχαριστήσω τον κ. Ιωάννη Ντουμπάκη για την επιπλέον προσφορά του στο εγχείρημα, όπως επίσης την οικογένειά μου και τα αγαπημένα μου πρόσωπα για την στήριξη και τη κατανόηση που μου έδειξε κατά τη περίοδο των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Η συγκεκριμένη πτυχιακή εργασία έχει να κάνει με τη θεωρητική ανάλυση, περιγραφή και προσομοίωση ενός επεξεργαστή μέσω της γλώσσας σχεδίασης ολοκληρωμένων συστημάτων VHDL. Γίνεται καθορισμός των παραμέτρων που απαιτούνται για τη λειτουργία του επεξεργαστή και παρουσιάζονται τα αποτελέσματα της σχεδίασης μέσω της χρήσης του πακέτου Altera Quartus.

ABSTRACT

The particular thesis concerns the theoretical analysis, functional description and simulation of a CPU by using VHDL. The appropriate parameters are determined and the simulation results are presented according to the Altera Quartus package.

Επιστημονική Περιοχή: Μικροηλεκτρονική

Λέξεις κλειδιά: οδηγία, execution, διακλάδωση, ALU, beq, MCU, FSM, Datapath, Execution, Instruction Fetch, Memory Writeback, Testbench, Package, Instruction Decode, MIPS

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

Ο νόμος του Moore προβλέπει ότι ο αριθμός των transistors που θα χρησιμοποιούνται σε ένα ολοκληρωμένο ψηφιακό κύκλωμα θα διπλασιάζεται ανά δεκαοκτώ μήνες. Ο εκθετικός αυτός ρυθμός ανάπτυξης καθιστά αναγκαία την επιστήμη που μελετά την οργάνωση και σχεδίαση μικροεπεξεργαστών για τη μέγιστη αξιοποίηση των πόρων που προέρχονται από τη τρέχουσα τεχνολογία. Η επιστήμη αυτή είναι γνωστή ως αρχιτεκτονική υπολογιστών.

Οι πρώτοι ψηφιακοί επεξεργαστές που δημιουργήθηκαν ανήκουν στην κατηγορία των CISC επεξεργαστών. Με τον όρο CISC (complex instruction set computer) αναφερόμαστε κυρίως στην αρχιτεκτονική συνόλου εντολών του εκάστοτε συστήματος, δηλαδή τις εντολές που υποστηρίζει. Οι CISC επεξεργαστές κάνουν χρήση εντολών κυμαινόμενου μήκους, σε bits, οι οποίες γενικά εκτελούνται σε αρκετούς κύκλους μηχανής. Μια εντολή CISC είναι μια μεγάλη σειρά από απλές λειτουργίες που απαρτίζουν μια περίπλοκη εντολή. Κλασικό παράδειγμα τέτοιας εντολής είναι το άθροισμα δύο διανυσμάτων αποθηκευμένων στη μνήμη σε ένα τρίτο διάνυσμα το οποίο αποθηκεύεται και αυτό με τη σειρά του στη κύρια μνήμη. Οι εντολές αυτές λόγω του κυμαινόμενου μήκους τους παρουσιάζουν ειδικά χαρακτηριστικά η κάθε μία και ο προγραμματιστής θα πρέπει να τα μεταχειρίζεται σα ξεχωριστές οντότητες. Έτσι ο προγραμματισμός σε ένα CISC επεξεργαστή είναι μία ιδιαίτερα δύσκολη και επίπονη διαδικασία που απαιτεί άριστη γνώση των επιμέρους λειτουργιών του συστήματος. Επιπλέον, λόγω της σύνθετης φύσης των εντολών αυτών απαιτείται μεγάλη περίοδος ρολογιού και άρα μικρή συχνότητα, κάτι που συμβάλλει αρνητικά στην απόδοση του συστήματος. Αντίθετα, σε ορισμένες περιπτώσεις αρχιτέκτονες που σχεδιάζουν CISC συστήματα έχουν την ευκαιρία για κατάλληλη συμπίεση των εντολών στη κρυφή μνήμη εντολών, εξοικονομώντας έτσι χώρο από τη μνήμη. Καθώς όμως τα transistors που χρησιμοποιούνται μικραίνουν σε μέγεθος, γεννήθηκαν δυσκολίες στις αρχιτεκτονικές αυτές καθώς δεν ήταν δυνατή η περεταίρω αξιοποίηση του ολοένα αυξανόμενου αριθμού των transistors.

Έτσι στη δεκαετία του 80’ ο David Patterson πρότεινε μια πολύ διαφορετική προσέγγιση στη μέχρι τότε παγιωμένη αγορά των CISC επεξεργαστών. Η εναλλακτική αυτή λογική τελικώς επικράτησε και ονομάστηκε RISC (reduced instruction set computing). Οι RISC επεξεργαστές είναι το αντίθετο των προκατόχων τους. Το σύνολο εντολών ενός τέτοιου επεξεργαστή περιλαμβάνει μόνο εντολές σταθερού και ίδιου μήκους. Έτσι, εξασφαλίζεται η συνοχή του συνόλου και η ευκολότερη κατανόησή του. Επιπλέον οι εντολές RISC είναι πολύ απλές βασικές λειτουργίες σε αντίθεση με τις περίπλοκες υλοποιήσεις των CISC. Έτσι, διακρίνονται απλές πράξεις πρόσθεσης, μεταφοράς δεδομένων από και προς τη μνήμη και συγκρίσεις. Επιπλέον η εκτέλεση κάθε εντολής ακολουθεί επικαλυπτόμενο μοντέλο, γεγονός που σημαίνει ότι η εκτέλεση της N+1 εντολής αρχίζει προτού να ολοκληρωθεί η

εκτέλεση της N εντολής. Η υλοποίηση αυτή δίνει τη δυνατότητα αύξησης της συχνότητας του ρολογιού επιτυγχάνοντας έτσι υψηλή απόδοση, έχοντας ταυτόχρονα απλουστευμένο σύνολο εντολών. Η κάθε λειτουργία ολοκληρώνεται σε λίγους κύκλους μηχανή λόγω της απλής φύσης της ενώ ο προγραμματισμός σε ένα RISC επεξεργαστή είναι σημαντικά ευκολότερος και αποδοτικότερος από έναν CISC.

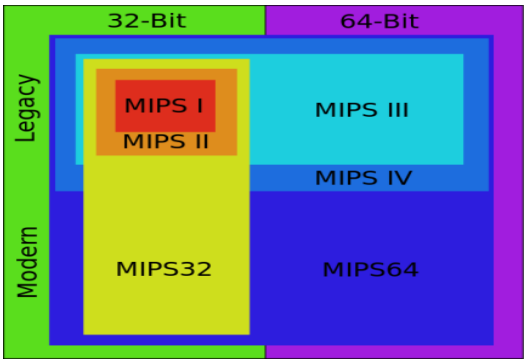
Για τους λόγους αυτούς δικαιολογείται η επικράτηση των RISC επεξεργαστών όπως και η συνεχόμενη βελτίωση και εξέλιξή τους.

Ο πρώτος επεξεργαστής MIPS ξεκίνησε από μια μελέτη της ομάδας του καθηγητή John L. Hennesy του Stanford University, το 1981. Η λογική ήταν το να αυξηθεί η απόδοση της λειτουργίας ενός υπολογιστικού συστήματος μέσω της χρήσης παράλληλων εντολών (instruction pipeline). Η συγκεκριμένη τεχνική ήταν γνωστή από πριν αλλά δεν είχε υλοποιηθεί πλήρως. Με τη χρήση της οι μηχανικοί είχαν πλέον τη δυνατότητα να διαχειρίζονται συστήματα που να μπορούν να υπερκαλύψουν κάποιες εντολές. Για παράδειγμα κατά τη διάρκεια που ένας υπολογιστής εκτελούσε μια μαθηματική πράξη η μνήμη μπορούσε να φορτώσει την επόμενη προς εκτέλεση εντολή, κάτι που δε γινόταν πρωτύτερα.

Ένα μεγάλο πρόβλημα ωστόσο, αποτελούσαν κάποιες συγκεκριμένες οδηγίες όπως η πράξη της διαίρεσης, οι οποίες χρειάζονταν περισσότερο χρόνο να εκτελεστούν, με αποτέλεσμα, ο επεξεργαστής να χρειαζόταν να περιμένει προτού να προχωρήσει στην επόμενη εντολή της ακολουθίας (pipelining). Η λύση βρισκόταν στη χρήση διασυνδεδεμένων μονάδων τα οποία να εκτελούν διαμοιραζόμενες λειτουργίες και βάσει μιας καθορισμένης προτεραιότητας. Πρακτικά όμως αυτό σήμαινε μεγάλη κατανάλωση χρόνου ρολογιού κάτι που αντίκρουε τη νοοτροπία του σχεδιασμού ενός MIPS το οποίο είχε σαν απαίτηση την πλήρη χρήση σε ένα κύκλο μηχανής. Επομένως η εξάλειψη αυτού του προβλήματος αποτέλεσε ένα επίπονο κομμάτι στην έρευνα του καθηγητή Hennesy.

Σε γενικές γραμμές ένα MIPS δεν αποτελεί παρά μια έκδοση ενός τυπικού RISC σχεδιασμού. Τα συστήματα αυτά περιορίζουν τον αριθμό των οδηγιών προς κωδικοποίηση με το να περιορίζουν κάποια bits οδηγίας. Χρησιμοποιούν 6 bits από τα 32 μιας λέξης για βασικό κωδικό λειτουργίας (opcode), με τα υπόλοιπα να αποτελούν είτε μια διεύθυνση άλματος (jump address) των 26-bits, είτε να περιλαμβάνουν μέχρι τέσσερα πεδία των 5-bits τα οποία να καθορίζουν μέχρι τρεις καταχωρητές συν μια τιμή ολίσθησης σε συνδυασμό με 6-bits κωδικού λειτουργίας. Αυτό επιτρέπει στη CPU να φορτώνει τις οδηγίες και τα δεδομένα που χρειάζεται σε ένα κύκλο μηχανής, όπου ένα διαφορετικό σύστημα όπως, για παράδειγμα, το MOS Technology 6502, χρειάζεται ξεχωριστούς κύκλους για να φορτώσει κωδικό λειτουργίας και δεδομένα.

Οι πρώτες MIPS αρχιτεκτονικές ήταν των 32-bit με εκδόσεις των 64-bit να προστίθενται αργότερα. Υπάρχουν πολλαπλές εκδόσεις των συστημάτων αυτών όπως οι MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32 και MIPS 64. Οι τωρινές εκδόσεις είναι οι MIPS32 (για 32-bit οδηγίες) και MIPS64 (για 64-bit οδηγίες). Οι εκδόσεις αυτές καθορίζουν ένα συγκεκριμένο καταχωρητή ελέγχου στο σύστημα όπως και το σετ οδηγιών. Το παρακάτω διάγραμμα μας δίνει ως παράδειγμα τη σχέση που έχουν οι διάφορες εκδόσεις μεταξύ τους.



Εικόνα 1.1 Σχέση εκδόσεων MIPS

Διάφορες επεκτάσεις των πακέτων έχουν δημιουργηθεί με το πέρασμα των χρόνων, όπως η MIPS-3D για οδηγίες τύπου SIMD και οι MIPS16e και MIPS MT για μικρότερη κατανάλωση μνήμης και πολυνηματικά περιβάλλοντα αντίστοιχα.

Υλοποιήσεις MIPS χρησιμοποιούνται κυρίως σε ενσωματωμένα(embedded) συστήματα όπως routers, κονσόλες video games(Nintendo 64, Play station, Play station 2 και PSP) καθώς και από εταιρείες του χώρου όπως οι Digital Equipment Corporation, NEC, Pyramid Technology, Siemens Nixdorf, Tandem Computers, SGI, Acer. Στα μέσα με τέλη της δεκαετίας του 1990 εκτιμάται ότι ένας στους τρεις μικροεπεξεργαστές RISC αποτελούσε μια υλοποίηση MIPS τύπου.

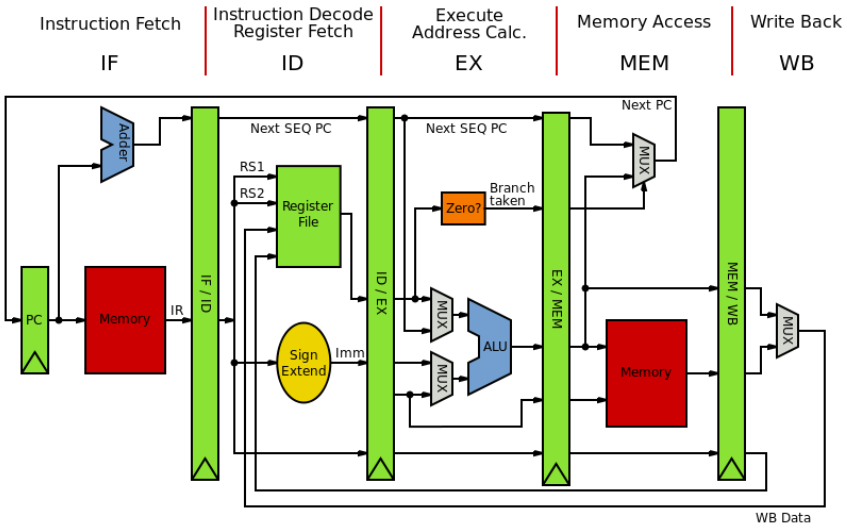
Η αρχιτεκτονική MIPS καθορίζει τη λειτουργία ενός επεξεργαστή έχοντας σαν βάση του διάφορους καταχωρητές. Άλλα είδη επεξεργαστών μπορούν να λειτουργούν με βάση τον σωρό (stack) ή τον συσσωρευτή (accumulator).

Το πρώτο μοντέλο MIPS κυκλοφόρησε το 1985 και ήταν το R2000. Εκτελούσε οδηγίες πολλαπλασιασμού και διαίρεσης, πολλαπλών κύκλων καταχωρώντας τα αποτελέσματα σε ένα προκαθορισμένο αρχείο (register file). Απαρτιζόταν από τριανταένα καταχωρητές γενικού σκοπού, των 32-bit και διέθετε καταχωρητή προγράμματος (program counter) ο οποίος δεν ήταν

άμεσα προσπελάσιμος. Οι οδηγίες που φορτώνονταν ήταν είτε τύπου big-endian είτε τύπου little-endian.

Το R2000 επίσης υποστήριζε μέχρι και τέσσερις διασυνδεδεμένους επεξεργαστές, ο ένας εκ των οποίων χρησιμοποιούταν για να διαχειρίζεται μνήμη, εξαιρέσεις στο κώδικα (exceptions) και σφάλματα, ενώ οι υπόλοιποι τρεις χρησιμοποιούνταν για άλλες λειτουργίες.

Το μοντέλο R3000, που εκδόθηκε το 1988, επέκτεινε το προηγούμενο προσθέτοντας 32kB cache μνήμης (instruction and data), ενώ το ύστερό του R4000, που εκδόθηκε το 1991, επέκτεινε την συγκεκριμένη αρχιτεκτονική σε μια πλήρη οργάνωση των 64-bit.



Εικόνα 1.2 Pipelined MIPS (instruction fetch, instruction decode, execute, memory access and write back)

Το υλικό εξελίχθηκε αργότερα μέχρι το R8000 (1994), που ήταν το πρώτο μοντέλο όπου μπορούσε να εκτελέσει δύο οδηγίες μνήμης σε ένα κύκλο και μέχρι το R16000 το οποίο έκανε χρήση μνήμης DDR SRAM ως cache και διέθετε αυξημένο κύκλο ρολογιού.

Αν και ο επεξεργαστής MIPS αποτέλεσε τη βάση για τα σύγχρονα επεξεργαστικά συστήματα, από το 1981 μέχρι σήμερα έγιναν πολλές αλλαγές στη θεώρηση της αρχιτεκτονικής των υπολογιστών. Νέες τεχνικές και ιδέες εισήλθαν στο χώρο αυτό ενώ παλαιότερες τεχνικές βελτιστοποιήθηκαν ή έπαψαν να χρησιμοποιούνται.

Πλέον η απόδοση των σύγχρονων συστημάτων εξαρτάται από την ικανότητα εκμετάλλευσης των τριών ειδών παραλληλισμού των εφαρμογών.

Έτσι διακρίνεται ο παραλληλισμός επιπέδου εντολής (instruction level parallelism - IPL), ο παραλληλισμός επιπέδου νήματος (thread level parallelism - TLP) και ο παραλληλισμός επιπέδου ροής δεδομένων (dataflow parallelism). Σε επίπεδο εντολής ο παραλληλισμός επιτυγχάνεται με αλγορίθμους για εκτέλεση εντολών εκτός σειράς, όπως ο αλγόριθμος Tomasulo. Με τη τεχνική αυτή δίνεται η δυνατότητα εκτέλεσης εντολών εκτός σειράς συμπληρώνοντας έτσι τις κενές θέσεις εκτέλεσης του επεξεργαστή, μεγιστοποιώντας τη χρησιμοποίηση των λειτουργικών του μονάδων. Η αξιοποίηση του ILP ήταν το πρώτο μεγάλο βήμα στη διαμόρφωση των επεξεργαστών όπως τους ξέρουμε σήμερα. Όμως πλέον εκτιμάται ότι η περεταίρω εκμετάλλευση του ILP είναι πολύ δύσκολη αν όχι αδύνατη καθώς οι τεχνικές που χρησιμοποιούνται γι' αυτό είναι βελτιστοποιημένες. Ένα ακόμη χαρακτηριστικό των σημερινών συστημάτων είναι η δυνατότητα παραγωγής και εκτέλεσης παράλληλου κώδικα. Έτσι ο προγραμματιστής ή πολλές φορές και ο ίδιος ο compiler κατακερματίζει μια εφαρμογή σε δύο ή περισσότερα κομμάτια ανεξάρτητα μεταξύ τους, που μπορούν να εκτελεστούν παράλληλα. Στη συνέχεια το κάθε κομμάτι εκτελείται σε διαφορετικό πυρήνα του επεξεργαστή ταυτόχρονα με τα υπόλοιπα. Το κάθε τέτοιο κομμάτι κώδικα ονομάζεται νήμα και η παράλληλη εκτέλεσή του υποστηρίζεται από πολυνηματικούς (multi-threaded) επεξεργαστές, ενώ ο τύπος παραλληλισμού που εμφανίζουν τέτοιες εφαρμογές ονομάζεται παραλληλισμός επιπέδου νήματος. Οι δύο αυτοί τύποι παραλληλισμού συναντώνται συχνά σε πολλές εφαρμογές και η εκμετάλλευσή τους είναι δυνατή από σχεδόν όλους τους σύγχρονους επεξεργαστές. Αντίθετα, ο παραλληλισμός επιπέδου ροής δεδομένων είναι πολύ δύσκολο να εντοπιστεί, να εξαχθεί και τελικώς να εκμεταλλευτεί από σύγχρονα συστήματα. Για το λόγο αυτό αποτελεί κύριο αντικείμενο έρευνας στη βιομηχανία των επεξεργαστών τη τελευταία δεκαετία. Η λογική στο παραλληλισμό επιπέδου ροής δεδομένων είναι η ικανότητα απεικόνισης της ροής των δεδομένων μέσα σε ένα σύστημα με ένα γράφο μετάβασης. Αν το σύστημα μπορεί να ακολουθήσει τη ροή αυτή, τότε ο τρόπος λειτουργίας του αλλάζει. Έτσι πλέον δεν εκτελεί εντολές απλά σε σειρά αλλά προσαρμόζεται στο τρόπο εκτέλεσης που του «επιβάλλει» η εκάστοτε εφαρμογή. Αυτό έχει επιτευχθεί σε ερευνητικό επίπεδο διατάσσοντας τις λειτουργικές μονάδες του επεξεργαστή στο χώρο με τέτοιο τρόπο ώστε να σχηματίζουν ένα δίκτυο διασύνδεσης (interconnection network) οργανωμένο σύμφωνα με συγκεκριμένα πρότυπα. Γνωστά συστήματα που εν μέρει εκμεταλλεύονται τον παραλληλισμό επιπέδου ροής δεδομένων είναι οι κάρτες γραφικών (graphics processing units) και εξυπηρετητές (servers) ειδικά διαμορφωμένοι για την εξυπηρέτηση μεγάλου όγκου πελατών. Όμως η πλήρης εκμετάλλευση του παραλληλισμού αυτού είναι ακόμη σε ερευνητικό επίπεδο και δεν έχει βρει αντίκτυπο σε βιομηχανικά συστήματα.

Αν και ένας σύγχρονος επεξεργαστής αξιολογείται ως προς την απόδοσή του με βάση τα παραπάνω κριτήρια, ο συνεχόμενα αυξανόμενος αριθμός των transistors που χωράνε σε ένα ολοκληρωμένο κύκλωμα δημιούργησε το πρόσφατο πρόβλημα της κατανάλωσης ισχύος. Αν και το θέμα αυτό ξεφεύγει από τα πλαίσια της συγκεκριμένης πτυχιακής εργασίας, αξίζει να αναφερθεί ότι σύγχρονα συστήματα υλοποιούν αλγορίθμους σε επίπεδο αρχιτεκτονικής και σε επίπεδο λογικής σχεδίασης για την μείωση της καταναλωμένης ενέργειας.

Η μελλοντική εξέλιξη των επεξεργαστών φαίνεται ότι θα ακολουθήσει μεγάλα άλματα στο διάστημα της ερχόμενης δεκαετίας. Με την εξέλιξη της τεχνολογίας υπάρχει ήδη νύξη για χρήση καινούριων υλικών εκτός του πυριτίου. Επιπλέον σε ερευνητικό επίπεδο οι κβαντικοί επεξεργαστές αποτελούν αντικείμενο έντονης έρευνας με πολλά ενθαρρυντικά αποτελέσματα. Το σίγουρο είναι ότι η επιστήμη αυτή θα συνεχίσει να χαρακτηρίζεται με συνεχείς αλλαγές γεννώντας νέα συστήματα και καινούριες αρχιτεκτονικές.

Εκτός από τους επεξεργαστές τύπου RISC ή CISC, αναπτύχθηκαν και άλλες αρχιτεκτονικές οι οποίες διέφεραν αρκετά από τους προκατόχους τους. Από αυτές τις πρωτοπόρες αρχιτεκτονικές αξίζει να αναφερθεί η VLIW (very long instruction word). Ο πρώτος επεξεργαστής τύπου VLIW που έκανε εμπορική επιτυχία ήταν ο Itanium, γνωστός και ως IA-64, ο οποίος αναπτύχθηκε μέσα από τη συνεργασία Intel και Hewllet – Packard. Η αρχιτεκτονική των VLIW επεξεργαστών διαφέρει σημαντικά από αυτή ενός CISC ή RIS. Αρχικά το μέγεθος εντολής είναι σημαντικά μεγάλο καθώς ανέρχεται σε 128 ή και περισσότερα bits, καθώς ο Itanium συνδυάζει πολλές μικρότερες RISC εντολές σε μία μακροεντολή. Με τον τρόπο αυτό δίνεται η δυνατότητα για δυναμική δρομολόγηση των εντολών προς εκτέλεση και γίνεται εκμετάλλευση του παραλληλισμού επιπέδου εντολής. Η απόδοση του συστήματος μπορεί να φτάσει σε πολύ υψηλά επίπεδα, αν η ομαδοποίηση αυτή των εντολών γίνει με βέλτιστο τρόπο. Το αρνητικό στους VLIW και κατ' επέκτασιν στον Itanium είναι ότι η προεργασία αυτή λόγω της εξαιρετικά περίπλοκης φύσης της δε γίνεται από το υλικό αλλά από το μεταγλωττιστή. Έτσι η διαδικασία αυτή γίνεται πολύ αργή και η τυχόν αδυναμία του μεταγλωττιστή για τη παραγωγή βελτιστοποιημένου κώδικα έχει πολύ μεγάλο αντίκτυπο στο χρόνο εκτέλεσης του προγράμματος. Λόγω του προβλήματος αυτού οι επεξεργαστές τύπου VLIW δεν εδραιώθηκαν στην αγορά, ενώ ο Itanium πλέον χρησιμοποιείται σε συστήματα εξυπηρετητών. Παρ' όλη την όχι και τόσο καλή του πορεία οι VLIW είναι άξιοι αναφοράς καθώς αντικατοπτρίζουν τη συνεχή αλλαγή και εξέλιξη της αρχιτεκτονικής υπολογιστών.

1.1 Περί επεξεργαστών

Η υλοποίηση του κάθε επεξεργαστή διαφέρει απ εταιρεία σε εταιρεία και από μοντέλο σε μοντέλο καθώς οι επιμέρους λεπτομέρειες της μικροαρχιτεκτονικής του κάθε συστήματος διαφοροποιούνται. όμως υπάρχουν ορισμένες σταθερές στη βιομηχανία των επεξεργαστών που αφορούν στο σχεδιασμό και τη λειτουργία των συγκεκριμένων κυκλωμάτων, οι οποίες μένουν απαράλλακτες. Παρακάτω παρουσιάζονται τέτοια κυκλώματα καθώς και οι αρχές λειτουργίας τους.

Ο φάκελος καταχωρητών (Register File) ενός επεξεργαστή είναι ένα σύνολο από flip-flops τα οποία δημιουργούνε ένα μικρό κύκλωμα αποθήκευσης πληροφορίας, άμεσα προσβάσιμο και πολύ υψηλής ταχύτητας. Συνήθως αποτελείται από 32 ή 64 καταχωρητές των 32 ή 64-bits ο καθένας, ανάλογα με την αρχιτεκτονική του συστήματος. Ωστόσο υπάρχουν και επεξεργαστές με 128 και πλέον καταχωρητές, αλλά αποτελούν εξαίρεση. Το κύκλωμα αυτό υλοποιείται σε όλα τα γνωστά συστήματα με ελάχιστες διαφορές από επεξεργαστή σε επεξεργαστή και συχνά χαρακτηρίζεται σα μια εσωτερική «κρυφή» μνήμη, πολύ μικρή και αρκετά γρήγορη. Στο MIPS η πρόσβαση του φακέλου καταχωρητών γίνεται στο δεύτερο και πέμπτο στάδιο υλοποίησης, τη πρώτη φορά για ανάγνωση και τη δεύτερη για εγγραφή αποτελεσμάτων αντίστοιχα.

Η κρυφή μνήμη (cache) είναι άλλο ένα βασικό κομμάτι του επεξεργαστή, αντίγραφο της οποίας υπάρχει σε κάθε σύγχρονο σύστημα. Η κρυφή μνήμη είναι και αυτή ένα κύκλωμα που αποτελείται από flip-flops, είναι πολύ μεγαλύτερο από αυτό του φακέλου καταχωρητών και τοποθετείται ιεραρχικά μεταξύ της μνήμης RAM και του φακέλου καταχωρητών. Χωρίζεται σε δύο μικρότερες υποκατηγορίες, τη κρυφή μνήμη εντολών (instruction cache) και τη κρυφή μνήμη δεδομένων (data cache). Στη μνήμη εντολών αποθηκεύονται οι εντολές προς εκτέλεση στον επεξεργαστή ενώ στη μνήμη δεδομένων γίνεται αποθήκευση των δεδομένων που παράγονται και καταναλώνονται από τη τρέχουσα εφαρμογή. Κύριο χαρακτηριστικό του κυκλώματος αυτού είναι η υψηλή ταχύτητα και το υψηλό κόστος. Για τη καλύτερη απόδοση και αξιοποίησή του αναπτύχθηκαν πολλές τεχνικές οργάνωσης της μνήμης ώστε να αυξάνεται το ποσοστό ευστοχίας και να μειώνεται το ποσοστό αστοχίας. Γενικότερα η μνήμη δεδομένων κατηγοριοποιείται σε τρία επίπεδα, τα L1, L2, L3 – cache. Όταν ζητείται η ανάγνωση ή η αποθήκευση δεδομένων που δεν υπάρχουν στους καταχωρητές, τότε γίνεται προσπέλαση στο επίπεδο L1 της κρυφής μνήμης. Αν η ζητούμενη τιμή δεν βρεθεί τότε έχουμε αστοχία της κρυφής μνήμης και γίνεται προσπέλαση στο επίπεδο L2 από όπου, σε αντίστοιχη αστοχία, γίνεται μετάβαση ελέγχου στο επίπεδο L3 και τέλος στη κύρια μνήμη RAM. Όταν εντοπιστεί η ζητούμενη πληροφορία τότε ακολουθείται η αντίστροφη διαδικασία για να φτάσει στον επεξεργαστή. Ο χρόνος που απαιτείται για τη συγκεκριμένη διαδικασία ονομάζεται ποινή

αστοχίας της κρυφής μνήμης. Για την περεταίρω βελτιστοποίηση της κρυφής μνήμης χρησιμοποιούνται διαφορετικές τεχνικές οργάνωσης όπως η άμεση απεικόνιση ή η συνολοσυσχετιστική μνήμη που όμως δε θα αναλυθούν περισσότερο καθώς ξεφεύγουν από τα πλαίσια της παρούσας εργασίας. Στον επεξεργαστή MIPS η προσπέλαση της μνήμης εντολών γίνεται στο πρώτο στάδιο της εντολής ενώ η προσπέλαση της μνήμης δεδομένων γίνεται στο τέταρτο στάδιο της εκτέλεσης.

Άλλο ένα ευρέως χρησιμοποιούμενο κύκλωμα είναι η αριθμητική και λογική μονάδα (ALU) που είναι υπεύθυνη για την εκτέλεση αριθμητικών και λογικών πράξεων όπως πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση, AND, OR και ούτω καθεξής. Αποτελείται από λογικές πύλες που χρησιμοποιούν τις τιμές των τάσεων των επιμέρους bits για να παράξουν το επιθυμητό αποτέλεσμα. Η οργάνωση, η δομή και η υλοποίηση της συγκεκριμένης μονάδας αποτέλεσε ένα θέμα που απασχόλησε για πολύ καιρό τους σχεδιαστές επεξεργαστών, για να καταλήξει να είναι πλέον πιο αποδοτική και γρήγορη ώστε να μην αποτελεί σημείο συμφόρησης (bottleneck) στη λειτουργία ενός συστήματος. Ο MIPS προβλέπει την αξιοποίηση της ALU στη Τρίτη φάση εκτέλεσης εντολής.

Το κύκλωμα υπεύθυνο για τη παραγωγή σημάτων που καθορίζουν διάφορες λειτουργίες του επεξεργαστή είναι γνωστό ως μονάδα ελέγχου. Η υλοποίηση της μονάδας αυτής διαφέρει από επεξεργαστή σε επεξεργαστή αλλά η λειτουργία της παραμένει ίδια. Η μονάδα αυτή παράγει τα σήματα επιλογής των πολυπλεκτών που οδηγούν τις εισόδους της ALU, τα σήματα επιλογής των πράξεων, επιλογής καταχωρητών, προσπέλασης μνήμης και γενικά έχει συντονιστικό ρόλο στις φάσεις εκτέλεσης εντολών. Η μονάδα ελέγχου αποτελείται από ένα κύκλωμα που υλοποιεί μια μηχανή πεπερασμένων καταστάσεων (FSM) η οποία είναι και ο πυρήνας λειτουργίας της. Σε παλαιότερους επεξεργαστές αντί του κυκλώματος αυτού υπήρχε μια μικρή μνήμη ανάγνωσης ROM η οποία περιείχε τη πληροφορία που απαιτούταν για την εκτέλεση της κάθε εντολής κώδικα. Το μοντέλο αυτό σύντομα εγκαταλείφθηκε στα σύγχρονα συστήματα ενώ χρησιμοποιείται μόνο σε ορισμένους επεξεργαστές κινητών τηλεφώνων ή για τν εκτέλεση εφαρμογών με χαμηλές απαιτήσεις ενέργειας.

1.2 Η μεταγλώττιση

Οι επεξεργαστές αποτελούνται από ψηφιακά ολοκληρωμένα κυκλώματα και η χρήση τους είναι η εκτέλεση κώδικα για την επεξεργασία δεδομένων και τη παραγωγή συγκεκριμένων αποτελεσμάτων. Στην ενότητα αυτή θα δούμε πώς ο κώδικας από μια γλώσσα προγραμματισμού υψηλού επιπέδου καταλήγει να εκτελείται στον επεξεργαστή.

Η μεταγλώττιση του κώδικα εξαρτάται από το μεταγλωττιστή (compiler) και τη πλατφόρμα που χρησιμοποιείται. Όμως υπάρχει ένα γενικό μοτίβο που ακολουθείται το οποίο είναι συνδεδεμένο με τις αρχές λειτουργίας των επεξεργαστών. Αρχικά γίνεται η μετάφραση του κώδικα σε μια ενδιάμεση αναπαράσταση χαμηλού επιπέδου από το μεταγλωττιστή. Στη συνέχεια ο κώδικας μετατρέπεται στη συμβολική γλώσσα assembly του εκάστοτε επεξεργαστή για να καταλήξει στο τέλος σε δυαδική μορφή, η οποία είναι και αναγνωρίσιμη από το σύστημα στο οποίο εκτελείται.

Η παραπάνω διαδικασία συμβαίνει γιατί τα ολοκληρωμένα κυκλώματα από τα οποία απαρτίζεται το σύστημα λειτουργούν με υψηλές και χαμηλές τάσεις, δηλαδή με λογικό μηδέν και ένα. Έτσι ένας επεξεργαστής δεν «καταλαβαίνει» ψευδοκώδικα assembly ή κάποια άλλη γλώσσα υψηλού επιπέδου. Για να γίνει η ανάθεση εντολών στο σύστημα απαιτείται η μετατροπή της γλώσσας αυτής στο δυαδικό σύστημα, το οποίο είναι και αναγνώσιμο από τον επεξεργαστή. Για το λόγο αυτό η γλώσσα που είναι πιο κοντά στην αρχιτεκτονική ενός επεξεργαστή είναι η assembly, η οποία δείχνει συμβολικά σε χαμηλό επίπεδο τις προς εκτέλεση εντολές, ενώ όμως παράλληλα είναι ιδιαίτερα δύσκολη και δε παρουσιάζει κάποιο επίπεδο αφαίρεσης.

1.3 Η γλώσσα VHDL

Η VHDL (VHSIC Hardware Description Language) είναι μια γλώσσα περιγραφής υλικού που χρησιμοποιείται στον αυτοματισμό ηλεκτρονικών σχεδιάσεων για τη περιγραφή μεικτών και ψηφιακών συστημάτων όπως FPGAs και ολοκληρωμένα κυκλώματα.

Αναπτύχθηκε στο Υπουργείο Άμυνας των Η.Π.Α. σα μια εναλλακτική λύση αντί της χρήσης μεγάλων και πολύπλοκων εγχειριδίων. Αρχικά υπήρξε η ανάπτυξη λογικών προσομοιωτών οι οποίοι θα μπορούσαν να διαβάσουν τα αρχεία VHDL με το επόμενο βήμα να βρίσκεται στην ανάπτυξη εργαλείων σύνθεσης που διάβαζαν τον κώδικα της VHDL και παρήγαγαν έναν ορισμό φυσικής υλοποίησης ενός κυκλώματος.

Η αρχική έκδοση της VHDL που αποτέλεσε το πρότυπο IEEE 1076-1987 περιλάμβανε αρκετούς τύπους δεδομένων όπως ακέραιους (integer), πραγματικούς (real), λογικούς (bit, boolean), χαρακτήρες (characters), χρόνο

(time) όπως και πίνακες αυτών (bit_vector, character). Το πρόβλημα αυτής της έκδοσης ήταν η χρήση της λογικής «πολλαπλών τιμών», κάτι που βελτιώθηκε με το πρότυπο IEEE 1164 το οποίο όριζε τύπους λογικής 9 τιμών (std_ulogic, std_ulogic_vector). Το πρότυπο IEEE 1076 έκανε τη σύνταξη πιο συνεπή ενώ άλλα θυγατρικά πρότυπα όπως το IEEE 1076.3 εισήγαγε προσημασμένους και μη προσημασμένους τύπους δεδομένων (signed/unsigned).

Η συγκεκριμένη γλώσσα χρησιμοποιείται συνήθως για τη συγγραφή μοντέλων που περιγράφουν ένα καθορισμένο λογικό κύκλωμα, σε κείμενο. Χρησιμοποιείται ένα πρόγραμμα προσομοίωσης για να δοκιμαστεί βάση μοντέλων προσομοίωσης, η λογική σχεδίαση ενός κυκλώματος. Η συλλογή αυτή μοντέλων προσομοίωσης που καθορίζονται στο πακέτο της γλώσσας ονομάζεται συχνά testbench. Η VHDL έχει δυνατότητα εισόδου - εξόδου αρχείων τα οποία χρησιμοποιούνται συνήθως από ένα testbench προσομοίωσης για το καθορισμό των διεγέρσεων ενός συστήματος (stimuli), την αλληλεπίδραση με το χρήστη και τη σύγκριση ληφθέντων – επιθυμητών δεδομένων.

Η σχεδίαση του υλικού μπορεί να γίνει σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Xilinx ISE, Altera Quartus) με σκοπό να παραχθεί το σχηματικό διάγραμμα RTL (Register Transfer Level) του επιθυμητού κυκλώματος και να επαληθευτεί μετά τη δημιουργία του κατάλληλου testbench, μέσω των κυματομορφών εισόδου – εξόδου του. Η δημιουργία ενός σωστού testbench προϋποθέτει σωστό ορισμό εισόδων.

Ένα βασικό πλεονέκτημα της συγκεκριμένης γλώσσας είναι ότι επιτρέπει τη περιγραφή (μοντελοποίηση) και επαλήθευση (προσομοίωση) ενός συστήματος, πριν τα εργαλεία σύνθεσης μεταφράσουν τη σχεδίαση σε πραγματικό υλικό. Ένα άλλο ατού της γλώσσας είναι ότι ένα έργο που δημιουργείται μπορεί να επιτρέψει τον ορισμό ταυτόχρονων συστημάτων (concurrent systems) και μπορεί να έχει πολλές εφαρμογές σε άλλα έργα λόγω του ότι είναι μεταφέρσιμο.

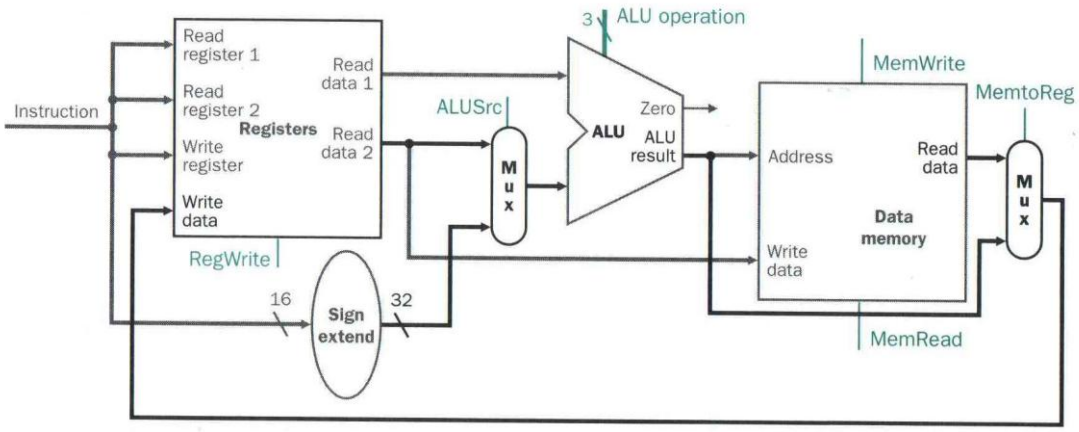
Στη VHDL η σχεδίαση αποτελείται από μία τουλάχιστον οντότητα (entity) που καθορίζει τη διεπαφή, μία αρχιτεκτονική (architecture) που καθορίζει τη πραγματική υλοποίηση (τι θα κάνει η οντότητα) και κάποιες βιβλιοθήκες σχεδίασης (library module). Ένα απλό παράδειγμα VHDL θα μπορούσε να είναι το παρακάτω:

```
--εισάγει την std_logic από τη βιβλιοθήκη της IEEE
	library ieee;
	use ieee.std_logic_1164.all;
--οντότητα
	Entity ANDGATE is
		Port( I1: in std_logic;
			I2: in std_logic;
			O: out std_logic);
--αρχιτεκτονική
	Architecture RTL of ANDGATE is
	Begin
			O<=I1 and I2;
	End architecture RTL;
```

Όταν ένα μοντέλο VHDL μεταφράζεται σε μία συστοιχία πυλών και γραμμών οι οποίες να αντιστοιχούν σε μια συσκευή όπως είναι ένα FPGA ή ένα CPLD, τότε το πραγματικό υλικό ρυθμίζεται και ο κώδικας δεν εκτελείται σε κάποιου επεξεργαστή

1.4 Ξεκινώντας μια απλή υλοποίηση

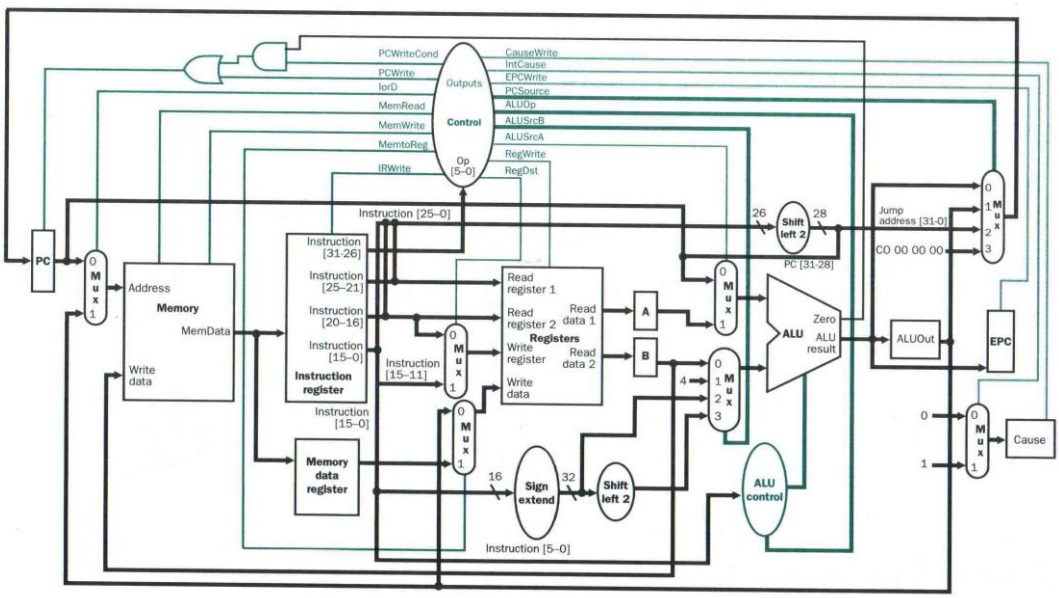
Αρχικά ξεκινάμε με μια απλή υλοποίηση ενός σχηματικού MIMP το βασικό υλικό και η συμπεριφορά του οποίου σχεδιάζονται με τη γλώσσα VHDL. Κατάλληλα τεστ θα επιβεβαιώσουν τη σωστή λειτουργία της ALU(or,and,sub,slt,add), την αναφορά στη μνήμη(load word, store word) και τις οδηγίες διακλάδωσης(beq, jump).



Εικόνα 1.3 Απλό μπλοκ διάγραμμα με κανάλια δεδομένων(datapaths)

1.5 Χρησιμοποιώντας υλοποιήσεις πολλαπλών κύκλων

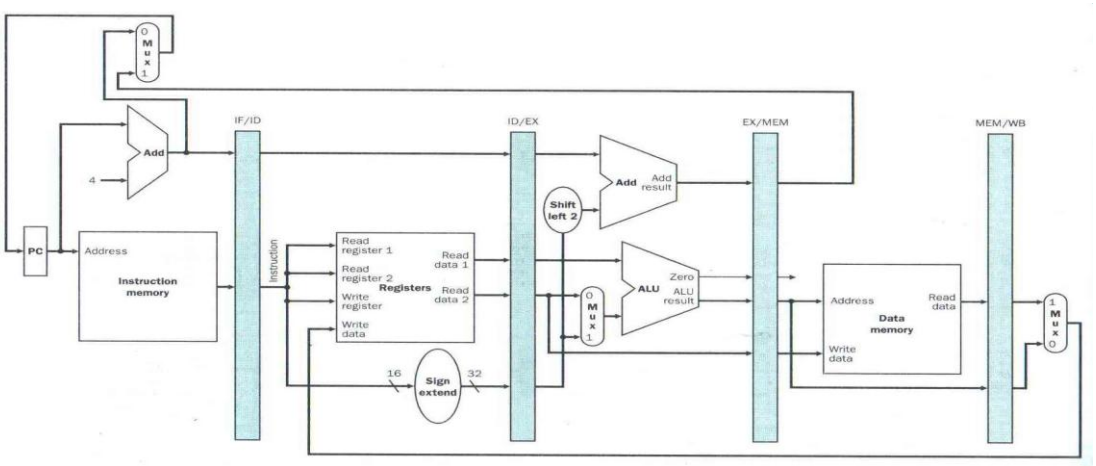
Λαμβάνοντας υπ' όψη ότι η αποδοτικότητα μιας μεγάλης υλοποίησης απλού κύκλου δεν θεωρείται και η καλύτερη δυνατή για την ταχύτητα χρήσης ενός επεξεργαστή, η βελτίωση επέρχεται με τη δημιουργία συστημάτων πολλαπλών κύκλων. Έτσι μπορούμε να έχουμε διαφορετικές τιμές κύκλων ρολογιού και οι λειτουργικές μονάδες μπορούν να μοιράζονται ανάλογα κατά την εκτέλεση.



Εικόνα 1.4 Σύστημα πολλαπλών κύκλων

1.6 Ενίσχυση απόδοσης με pipelining

Με σκοπό την ενίσχυση της απόδοσης ενός συστήματος επεξεργαστή καθώς και την αύξηση της ταχύτητας υπολογισμών του, μια καλή υλοποίηση γίνεται με τη βοήθεια της τεχνικής της σωλήνωσης(pipelining). Πολλαπλές εντολές επικαλύπτονται κατά τη διάρκεια της εκτέλεσης ούτως ώστε κάποια στάδια να δουλεύουν εκ παραλλήλου.



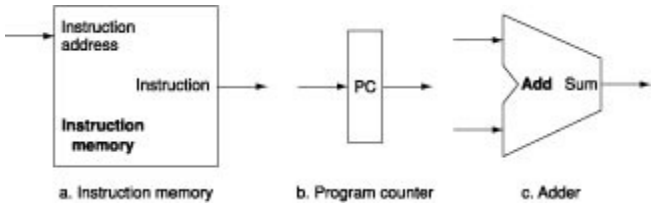
Εικόνα 1.5 Δομή pipeline

2. ΠΡΟΣΔΙΟΡΙΣΜΟΣ ΣΤΟΧΟΥ

2.1 Δημιουργία ενός καναλιού δεδομένων (datapath)

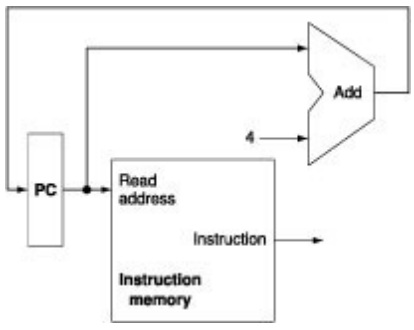
2.1.1 Βασικά εξαρτήματα

Πρώτα ρίχνουμε μια ματιά στα στοιχεία που απαιτούνται για την εκτέλεση εντολών και στις συνδέσεις τους. Το πρώτο που χρειαζόμαστε είναι χώρος για την αποθήκευση του προγράμματος. Αυτή η μνήμη εντολών (Instruction Memory) ελέγχει τις εντολές μέσω καθορισμένων διευθύνσεων τις οποίες προωθεί σε ένα μετρητή (Program Counter) και μέσω και ενός αθροιστή (Adder), αυξάνει το μετρητή ώστε να δείχνει τη διεύθυνση της επόμενης εντολής. Όλα αυτά τα στοιχεία φαίνονται παρακάτω.



Εικόνα 2.1 Μνήμη, Μετρητής, Αθροιστής

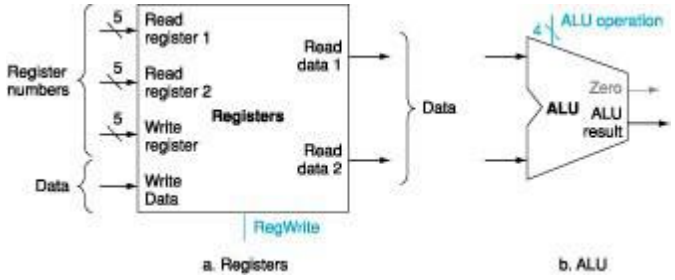
Με το τέλος της εκτέλεσης μιας εντολής ο PC (Program Counter) αυξάνεται κατά 4 bytes ώστε να δείχνει στη διεύθυνση της επόμενης προς εκτέλεση εντολής. Αυτό γίνεται περισσότερο κατανοητό από την επόμενη εικόνα.



Εικόνα 2.2 Fetching μνήμης

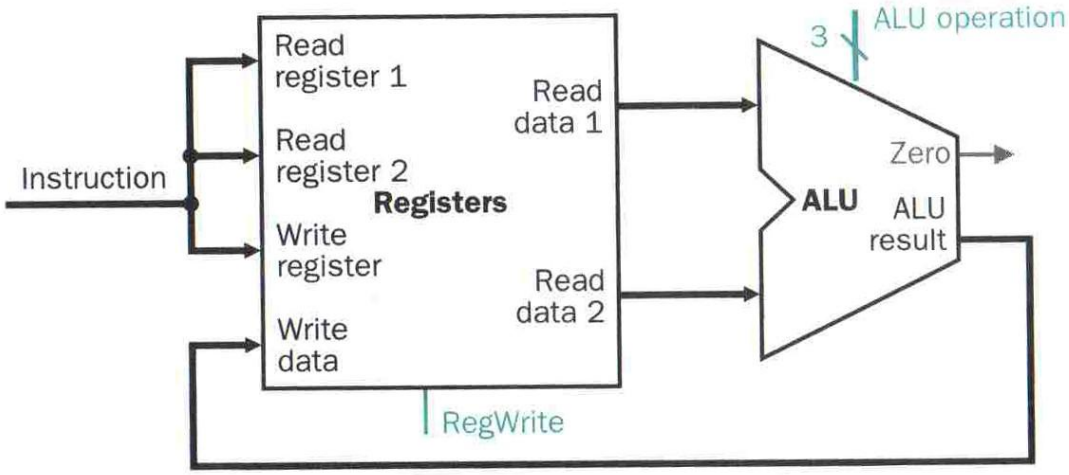
2.1.2 Στοιχεία αριθμητικών και λογικών συναρτήσεων

Οι εντολές χειρίζονται δυο καταχωρητές, εκτελούν μια λειτουργία ALU και επιστρέφουν το αποτέλεσμα. Εδώ εκτελούνται οι πράξεις add,or,and,slt,sub. Οι 32 καταχωρητές του επεξεργαστή κρατούνται σε ένα καθορισμένο αρχείο (Register File). Για ανάγνωση μιας λέξης δεδομένων (word) χρειάζονται δυο είσοδοι των 5 bits οι οποίοι καθορίζουν τον αριθμό του καταχωρητή που πρόκειται να διαβαστεί και δυο έξοδοι των 32 bits οι οποίες περιέχουν τη τιμή αυτού. Το επιστρεφόμενο αποτέλεσμα κάνει χρήση δυο εισόδων, μια που καθορίζει τον αριθμό του καταχωρητή και μια που προωθεί τα δεδομένα προς εγγραφή. Περισσότερα για τους καταχωρητές και τη διευθυνσιοδότηση εντολών θα δούμε σε επόμενο κεφάλαιο.



Εικόνα 2.3 Καταχωρητές, ALU

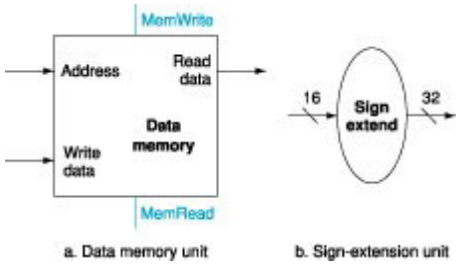
Για την επεξεργασία των δεδομένων των καταχωρητών απαιτείται μια μονάδα ALU δυο εισόδων με τη παρακάτω σύνδεση.



Εικόνα 2.4 Οδηγίες ALU

2.1.3 Φόρτωση και αποθήκευση λέξης (lw, sw)

Δυο επιπλέον στοιχεία χρειάζονται για την υλοποίηση των εντολών lw και sw. Αυτά είναι η μνήμη δεδομένων (Data Memory) και η μονάδα SEU (Signed Extension Unit).

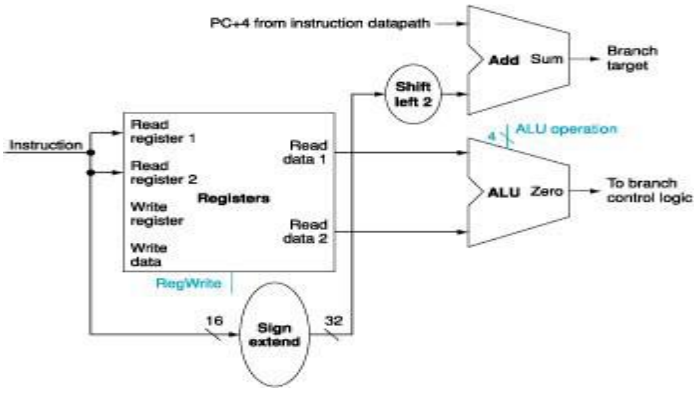


Εικόνα 2.5 Μνήμη δεδομένων, SEU

Οι εντολές lw, sw υπολογίζουν μια διεύθυνση μνήμης προσθέτοντας τη τιμή ενός καταχωρητή στο 16-bit offset που περιέχεται στην συγκεκριμένη εντολή.

2.1.4 Εντολή branch on equal (beq)

Η συγκεκριμένη εντολή χρησιμοποιεί δυο καταχωρητές που συγκρίνονται ως προς την ισότητά τους και ένα offset των 16-bits που χρησιμοποιείται για να υπολογίσει τη διεύθυνση στόχου της διακλάδωσης (branch) σε σχέση με την αρχική διεύθυνση της εντολής.



Εικόνα 2.6 Δομή διακλάδωσης

Η συγκεκριμένη δομή συγκρίνει τους καταχωρητές και υπολογίζει τον στόχο. Το πεδίο διευθύνσεων της εντολής διακλάδωσης πρέπει να έχει μέγεθος 16

έως 32 bits και να υποστεί αριστερή ολίσθηση κατά 2 bits έτσι ώστε να αποτελεί ένα offset λέξης. Η διεύθυνση του στόχου διακλάδωσης υπολογίζεται προσθέτοντας τη διεύθυνση της επόμενης εντολής (PC+4) με το προηγούμενο offset.

2.1.5 Η εντολή jump

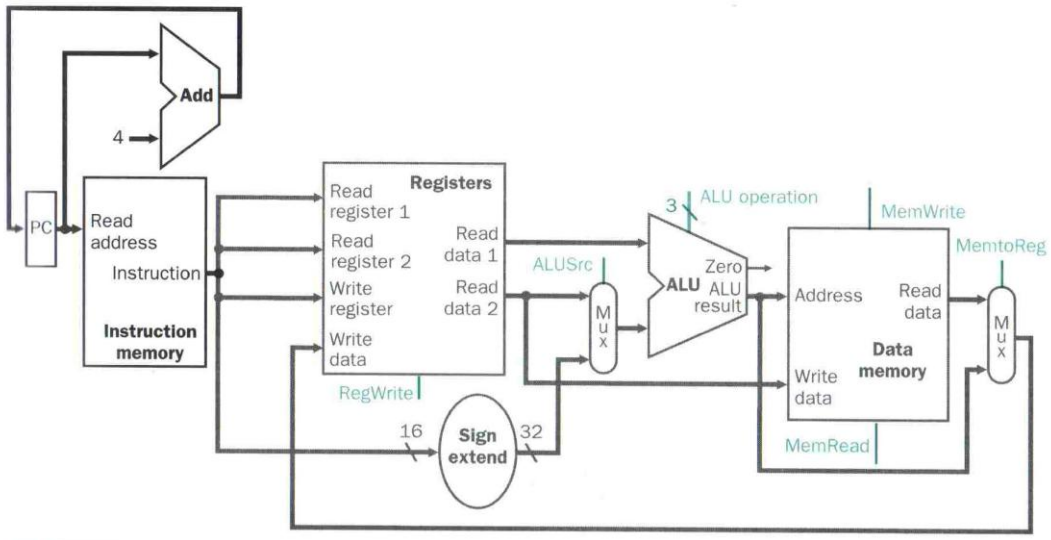
Η εντολή αναπήδησης (jump) υπολογίζεται λίγο διαφορετικά από αυτή της διακλάδωσης. Η διεύθυνση προορισμού μιας αναπήδησης δημιουργείται συνενώνοντας τα 4 πιο σημαντικά bits της τρέχουσας τιμής PC+4 με το 26-bits πεδίο δεδομένων της συγκεκριμένης εντολής και αλλάζοντας τα δυο τελευταία bits σε ‘00’.

2.2 Απλή υλοποίηση MIPS

Η πιο απλή υλοποίηση ενός τέτοιου συστήματος που μπορούμε να έχουμε είναι ο συνδυασμός όλων των παραπάνω στοιχείων με χρήση επιπρόσθετων γραμμών ελέγχου.

2.2.1 Δημιουργώντας μια μονή διαδρομή δεδομένων

Η απλούστερη διαδρομή (datapath) πιθανώς να επιχειρήσει την εκτέλεση όλων των εντολών σε ένα κύκλο ρολογιού. Αυτό σημαίνει πως κάθε στοιχείο μπορεί να χρησιμοποιηθεί μόνο μια φορά για κάθε εντολή. Τα συγκεκριμένα στοιχεία μπορούν να διαμοιραστούν ανάμεσα σε διαφορετικές ροές εντολών. Υπάρχει η δυνατότητα χρήσης πολλαπλής εισόδου μέσω πολυπλέκτη. Η παρακάτω εικόνα παρουσιάζει ένα πιο ολοκληρωμένο συνδυασμό των μνημών με την ALU και τους σχετικούς καταχωρητές.



Εικόνα 2.7 Ολοκληρωμένο σύστημα απλής διαδρομής με χρήση πολυπλεκτών

2.2.2 Έλεγχος ALU

Εδώ μας είναι απαραίτητη η χρήση του πεδίου MIPS το οποίο φαίνεται παρακάτω με όλες τις πληροφορίες του.

Op	Rs	Rt	Rd	Shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

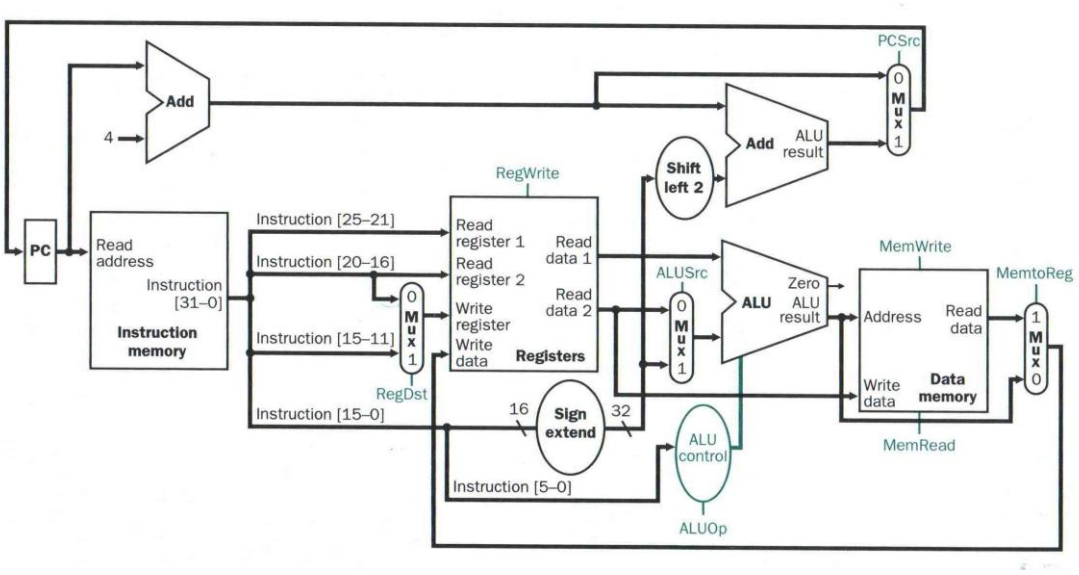
Με

- Op = Κωδικός λειτουργίας της εντολής
- Rs = Πρώτος καταχωρητής πηγής
- Rt = Δεύτερος καταχωρητής πηγής
- Rd = Καταχωρητής προορισμού
- Shamt = Ποσό ολίσθησης(shift amount)
- Funct = Καθορίζει τη λειτουργία που θα εκτελεστεί από την εντολή σύμφωνα με το πεδίο op

Στο σχήμα που ακολουθεί βλέπουμε στη τελευταία στήλη τα 3-bit για τον έλεγχο εισόδου ALU ο οποίο εξαρτάται από τα 2-bit του σήματος ALUop, που ενεργοποιείται από τη μονάδα κύριας λειτουργίας MCU(Main Control Unit), όπως και από τα 6-bit του πεδίου funct από την εντολή MIPS.

Op	**ALUop**	**Οδηγία λειτουργίας**	**funct**	**Προσδοκώμενη ενέργεια ALU**	**Είσοδος ελέγχου ALU**
LW	00	Load word	XXXXXX	Add	010
SW	00	Store word	XXXXXX	Add	010
beq	01	branch equal	XXXXXX	Subtract	110
R-type	10	Add	100000	Add	010
R-type	10	Subtract	100010	Subtract	110
R-type	10	And	100100	And	000
R-type	10	Or	100101	Or	001
R-type	10	set on less than	101010	Set on less than	111

Πίνακας 2.1 Έλεγχος ALU



Εικόνα 2.8 Χρήση μονάδας ελέγχου ALU

2.2.3 Μονάδα κύριας λειτουργίας MCU

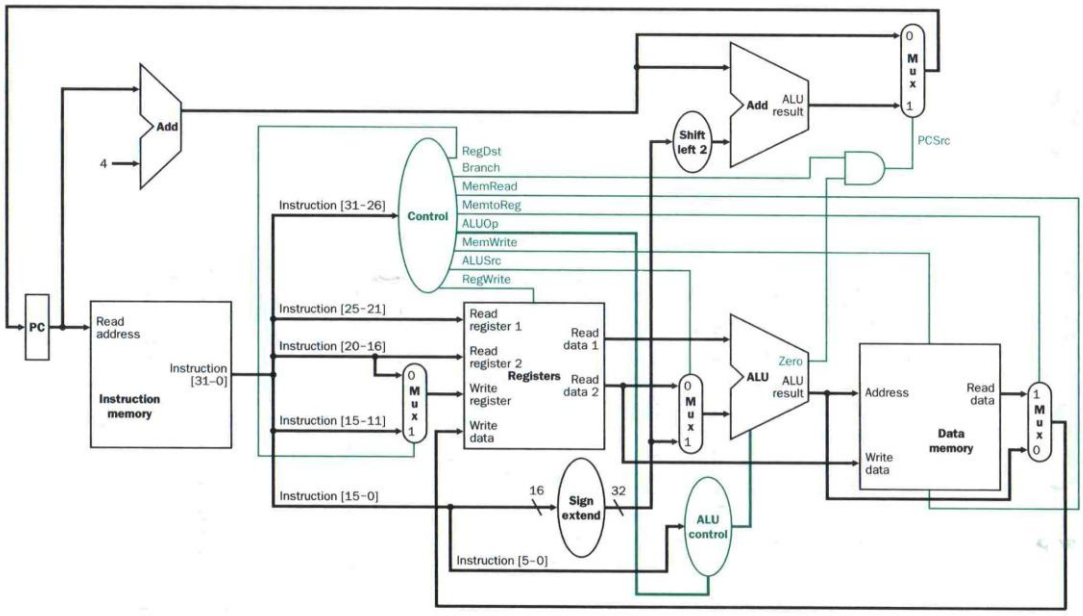
Η μονάδα κύριας λειτουργίας ενεργοποιεί τα bits ελέγχου για τους πολυπλέκτες, τη μνήμη δεδομένων και την ALU. Η είσοδος της μονάδας αυτής είναι το 6-bit πεδίο op της οδηγίας MIPS.

Στον επόμενο πίνακα βλέπουμε την έννοια των διαφόρων σημάτων ελέγχου.

Όνομα σήματος	Όχι σε χρήση (deassserted)	Σε χρήση (asserted)
RegDst	Ο αριθμός του καταχωρητή προορισμού για το καταχωρητή εγγραφής έρχεται από το πεδίο rt	Ο αριθμός του καταχωρητή προορισμού για το καταχωρητή εγγραφής έρχεται από το πεδίο rd
RegWrite	-	Στην είσοδο Write Register των κατχωρητών εγγράφεται η τιμή Write Data της εισόδου της μνήμης δεδομένων
ALUSrc	Ο δεύτερος τελεστής για την ALU έρχεται από το δεύτερο αρχείο καταχωρητών εξόδου(Read data 2)	Ο δεύτερος τελεστής για την ALU είναι τα 16 λιγότερο σημαντικά bits της οδηγίας, με επέκταση προσήμου
PCSrc	Το PC αντικαθίσταται από την έξοδο του αθροιστή που υπολογίζει τη τιμή PC +4	Το PC αντικαθίσταται από την έξοδο του αθροιστή που υπολογίζει το στόχο εναλλαγής (branch target)
MemRead	-	Τα δεδομένα που ορίζονται στην είσοδο Address τοποθετούνται στην έξοδο Read data
MemWrite	-	Τα δεδομένα που ορίζονται στην είσοδο Address αντικαθιστώνται από τη τιμή στην είσοδο Write data
MemtoReg	Η τιμή που πηγαίνει στην είσοδο Write data των καταχωρητών έρχεται από την ALU	Η τιμή που πηγαίνει στην είσοδο Write data των καταχωρητών έρχεται από τη μνήμη δεδομένων

Πίνακας 2.2 Σήματα ελέγχου

Η σύνδεση της μονάδας ελέγχου παρουσιάζεται στο επόμενο σχήμα.



Εικόνα 2.9 Απλή διαδρομή με σύνδεση μονάδας ελέγχου MCU

Συνδυάζοντας τις εικόνες 2.9 και 2.8 οδηγούμαστε στον παρακάτω πίνακα αληθείας της MCU

Οδηγία	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALU Op1	ALU Op2
R-format	1	0	0	1	0	0	0	1	0
Lw	0	1	1	1	1	0	0	0	0
Sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Πίνακας 2.3 Πίνακας αληθείας της μονάδας ελέγχου

2.2.4 Μειονεκτήματα της υλοποίησης μονού κύκλου

Στις νέες τεχνολογίες η χρήση μιας διάταξης επεξεργαστή μονού κύκλου δεν συνηθίζεται λόγω της αναποτελεσματικότητάς της. Ένας κύκλος ρολογιού πρέπει να έχει το ίδιο μήκος για κάθε εντολή και ως εκ τούτου καθορίζεται από το μεγαλύτερο δυνατό μονοπάτι. Αυτό είναι σχεδόν το μονοπάτι της εντολής load word (lw) που χρησιμοποιεί πέντε μονάδες στη σειρά, τη μνήμη εντολών, το αρχείο καταχωρητών, την ALU, τη μνήμη δεδομένων και πάλι το αρχείο καταχωρητών.

Οι διατάξεις μονού κύκλου υφίστανται μόνο σε περιπτώσεις όπου έχουμε ένα πολύ μικρό κομμάτι εντολών προς εκτέλεση.

2.3 Υλοποίηση πολλαπλού κύκλου

Με σκοπό να αποφύγουμε τα μειονεκτήματα της χρήσης μονού κύκλου που περιγράφηκε στο προηγούμενο εδάφιο, οργανώνουμε σύστημα πολλαπλού κύκλου.

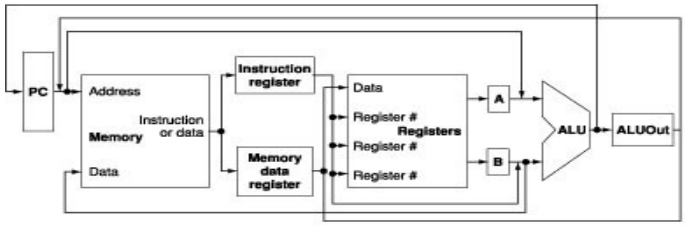
Η τεχνική αυτή διαιρεί τη κάθε εντολή σε βήματα καθένα από τα οποία εκτελείται σε ένα κύκλο ρολογιού.

Εδώ μπορεί μια λειτουργική μονάδα να δουλέψει παραπάνω από μία φορές μέσα στην ίδια εντολή, κάτι που μας επιτρέπει την εξοικονόμησή τους.

Μεγάλο πλεονέκτημα της υλοποίησης αυτής αποτελεί το γεγονός ότι μπορούμε να διαμοιράσουμε μονάδες κατά την εκτέλεση εντολών.

2.3.1 Προσθήκες και αλλαγές στο σχηματικό

Στη παρακάτω εικόνα βλέπουμε με μια πρώτη ματιά τη γενική δομή ενός μοντέλου πολλαπλού κύκλου.



Εικόνα 2.10 Μοντέλο πολλαπλού κύκλου

Οι διαφορές με το μοντέλο μονού κύκλου είναι ότι εδώ χρησιμοποιείται μόνο μία μονάδα μνήμης για εντολές και δεδομένα, μόνο μία σκέτη μονάδα ALU χωρίς χρήση αθροιστών και επιπλέον καταχωρητές εξόδου με σκοπό να διατηρούν τη τιμή εξόδου μέχρις ότου γίνει χρήση αυτής σε κάποιο μετέπειτα κύκλο.

Όπως βλέπουμε στο σχήμα ο καταχωρητής εντολών (instruction register → IR) και ο καταχωρητής μνήμης δεδομένων (Memory Data Register → MDR) αποθηκεύουν την έξοδο της μνήμης. Οι καταχωρητές A και B αποθηκεύουν τα στοιχεία που διαβάζονται στο αρχείο καταχωρητών και η ALUOut κρατά την έξοδο της ALU.

Με εξαίρεση τον IR όλοι αυτοί οι καταχωρητές διατηρούν τα δεδομένα τους μόνο κατά το διάστημα μεταξύ δύο παρακείμενων παλμών ρολογιού. Ο IR διατηρεί τη τιμή του σε όλη τη διάρκεια εκτέλεσης μιας εντολής οπότε και χρειάζεται ένα κατάλληλο σήμα ελέγχου (write control signal).

Η χρήση μίας μόνο μονάδας ALU προκαλεί τις εξής αλλαγές στο μονοπάτι:

- Ένας επιπλέον πολυπλέκτης προστίθεται στην πρώτη είσοδο της ALU με σκοπό το να επιλέξει μεταξύ του καταχωρητή A και του PC
- Ο πολυπλέκτης στη δεύτερη είσοδο της ALU αλλάζει από δύο σε τεσσάρων καταστάσεων. Οι δύο νέες είσοδοι είναι η σταθερά 4 για αλλαγή του PC και η ολισθημένη και με επέκταση προσήμου τιμή του πεδίου offset της εντολής διακλάδωσης (branch instruction).

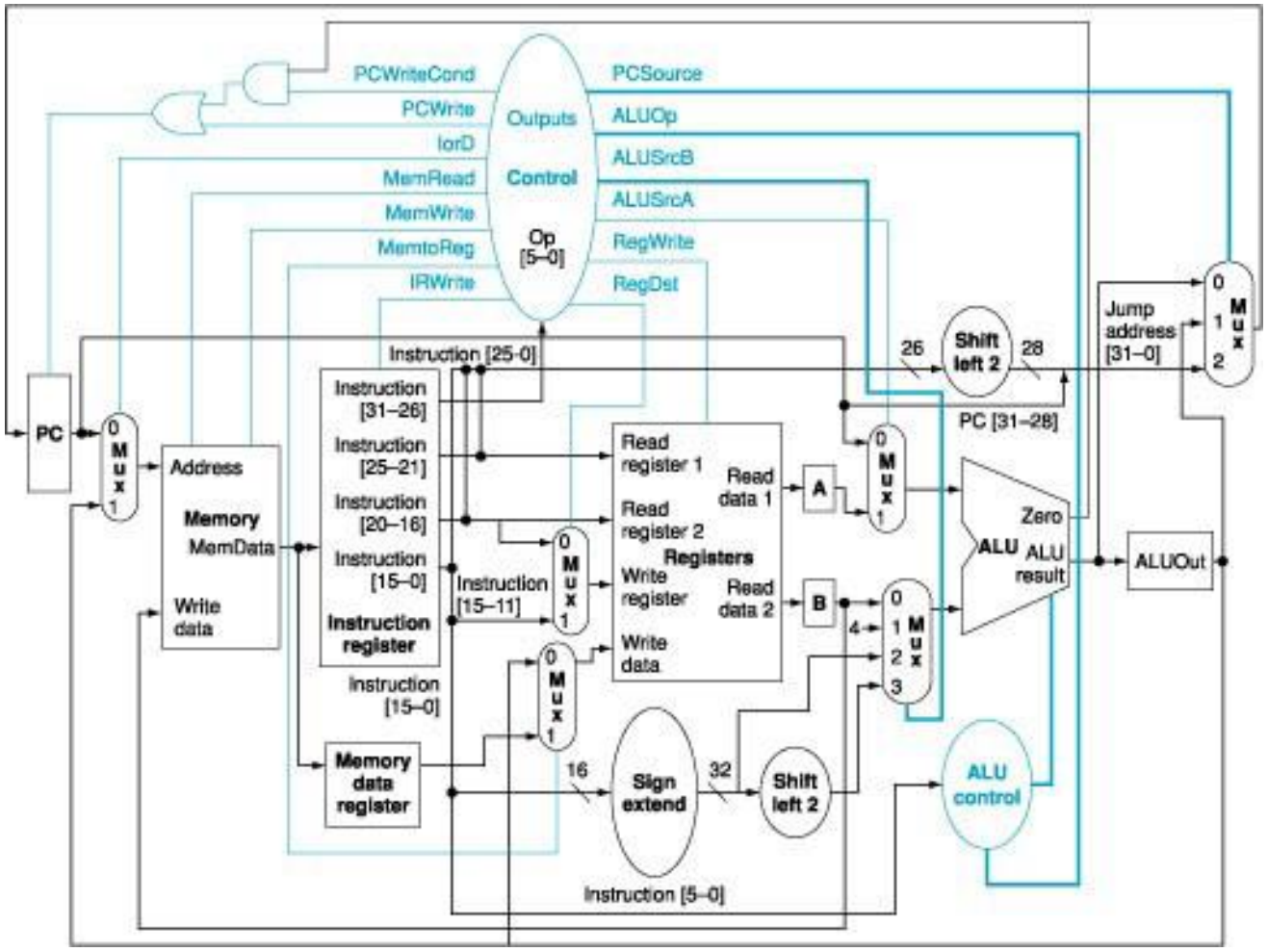
Με σκοπό να μπορούμε να ελέγξουμε τυχόν περιπτώσεις διακλάδωσης ή άλματος, απαιτούνται κάποιες προσθήκες στο μονοπάτι.

Οι περιπτώσεις οδηγιών τύπου-R, οδηγιών διακλάδωσης και άλματος δημιουργούν τρείς τιμές προς εγγραφή στον PC.

- Η έξοδος της ALU η οποία είναι PC +4
- Η τιμή του ALUOut μετά τον υπολογισμό της διεύθυνσης στόχου της διακλάδωσης (branch target address)
- Τα 26 λιγότερο σημαντικά bits του IR ολισθημένα αριστερά κατά 2 και συνενωμένα με τα 4 περισσότερο σημαντικά bits της προσαυξημένης τιμής του PC, όταν πρόκειται για οδηγία άλματος (jump).

Ειδικότερα ο PC χρειάζεται δύο σήματα ελέγχου(write signals) ένα PCWrite για συνεχή εγγραφή (PC +4 ή οδηγία άλματος - jump) και ένα PCWriteCond για συνεχή εγγραφή (εντολή διακλάδωσης - branch).

Το παρακάτω σχήμα δείχνει μια πιο ολοκληρωμένη δομή πολλαπλού κύκλου. Παρατηρούμε πως το σήμα ελέγχου για τον PC αποτελεί ένα συνδυασμό μέσω κατάλληλων πυλών AND, OR των σημάτων Zero της ALU και των PCWrite και PCWriteCond.



Εικόνα 2.11 Ολοκληρωμένο μονοπάτι πολλαπλού κύκλου.

2.3.2 Εκτέλεση οδηγιών σε κύκλους ρολογιού

Η εκτέλεση μιας εντολής διαιρείται σε κύκλους ρολογιού, κάτι που σημαίνει πως η κάθε οδηγία ακολουθεί μια καθορισμένη σειρά βημάτων. Στους παρακάτω πίνακες τοποθετούνται όλα τα σήματα ελέγχου με τις λειτουργίες τους.

Όνομα σήματος	**Σήμα απενεργοποιημένο**	**Σήμα ενεργοποιημένο**
RegDst	Ο αριθμός προορισμού για τον καταχωρητή Write έρχεται από το πεδίο rt	Ο αριθμός προορισμού για τον καταχωρητή Write έρχεται από το πεδίο rd
RegWrite	-	Ο καταχωρητής γενικού σκοπού που επιλέγεται από τον αριθμό του καταχωρητή Write εγγράφεται μαζί με τη τιμή της εισόδου Write data
ALUSrcA	Το πρώτο στοιχείο της ALU είναι ο PC	Το πρώτο στοιχείο της ALU έρχεται από το καταχωρητή A
MemRead	-	Το περιεχόμενο που δείχνει η είσοδος address της μνήμης γράφεται στην έξοδο Memory data
MemWrite	-	Το περιεχόμενο που δείχνει η είσοδος address της μνήμης αντικαθίσταται από τη τιμή της εισόδου Write data
MemtoReg	Η τιμή που πηγαίνει στην είσοδο Write data των καταχωρητών έρχεται από το ALUOut	Η τιμή της εισόδου Write data έρχεται από τον MDR
IorD	Ο PC στέλνει τιμή στην είσοδο address της μνήμης	Το ALUOut στέλνει τιμή στην είσοδο address της μνήμης
IRWrite	-	Η έξοδος της μνήμης γράφεται στο IR
PCWrite	-	Έχει γίνει εγγραφή του PC. Η πηγή ελέγχεται από το PCSource
PCWriteCond	-	Έχει γίνει εγγραφή του PC εάν η έξοδος Zero

		της ALU είναι επίσης ενεργοποιημένη

Πίνακας 2.4 Ενέργειές των σημάτων ελέγχου 1-bit

Όνομα σήματος	**Τιμή**	**Ενέργεια**
ALUOp	00	Η ALU κάνει πρόσθεση
	01	Η ALU κάνει αφαίρεση
	10	Το πεδίο function της οδηγίας καθορίζει τη λειτουργία που θα εκτελέσει η ALU
ALUSrcB	00	Η δεύτερη είσοδος της ALU έρχεται από το καταχωρητή B
	01	Η δεύτερη είσοδος της ALU είναι η σταθερά 4
	10	Η δεύτερη είσοδος της ALU είναι τα 16 λιγότερο σημαντικά bits του IR με επέκταση προσήμου
	11	Η δεύτερη είσοδος της ALU είναι τα 16 λιγότερο σημαντικά bits του IR με επέκταση προσήμου και ολισθημένα αριστερά κατά 2
PCSource	00	Η έξοδος της ALU(PC +4) στέλνεται στον PC για εγγραφή
	01	Το περιεχόμενο του ALUOut(διεύθυνση στόχου διακλάδωσης) στέλνεται στον PC για εγγραφή
	10	Η διεύθυνση στόχου άλματος(IR[25-0] ολισθημένα αριστερά κατά 2 και συνενωμένα με τα PC +4[31-28]) στέλνεται

		στον PC για εγγραφή

Πίνακας 2.5 Ενέργειες των σημάτων ελέγχου 2-bit

Η εκτέλεση μιας εντολής χωρίζεται σε 5, το πολύ, βήματα. Διαφορετικά στοιχεία μπορούν να λειτουργήσουν παράλληλα μέσα σε ένα κύκλο ρολογιού. Πρέπει να γίνει σαφές ότι μετά από ένα συγκεκριμένο βήμα οι υπολογισμένες τιμές βρίσκονται αποθηκευμένες είτε στη μνήμη, είτε σε κάποιο καταχωρητή. Μια περισσότερο θεωρητική προσέγγιση σχετικά με τα βήματα που ακολουθεί κατά την εκτέλεση εντολών ο MIPS θα δούμε παρακάτω. Πρακτικά τα βήματα αυτά είναι:

1. Fetch εντολής
   Fetch από μνήμη και υπολογισμός διεύθυνσης εντολής

   IR = Memory [PC]
   PC = PC + 4, με τα σήματα ελέγχου να παίρνουν τις τιμές

   MemRead = 1
   IRWrite = 1
   IorD = 0
   ALUSrcA = 1
   ALUSrcB = 01
   ALUOp = 00
   PCSource = 00
   PCWrite = 1

2. Αποκωδικοποίηση εντολής και fetch καταχωρητών
   Ακόμα δεν είναι γνωστό ποια είναι η ακριβής εντολή οπότε εκτελούνται ενέργειες εφαρμόσιμες για κάθε εντολή.
   Οι καταχωρητές που υποδεικνύονται στα πεδία rs και rd της εντολής διαβάζονται και αποθηκεύονται στους A και B. Ο ενδεχόμενος στόχος διακλάδωσης υπολογίζεται και καταχωρείται στο ALUOut.

   A = Reg[IR[25-21]]
   B = Reg[IR[20-16]]
   ALUOut = PC + (sign-extend (IR[15-0]) << 2), με τα σήματα ελέγχου να παίρνουν τις τιμές

   ALUSrcA = 0
   ALUSrcB = 11
   ALUOp = 00

3. Υπολογισμός διεύθυνσης μνήμης ή συμπλήρωση διακλάδωσης

   Εδώ είναι γνωστή η εντολή και έχουμε τέσσερις συγκεκριμένες λειτουργίες.

   Α) Αναφορά στη μνήμη:

   ALUOut = A + sign-extend(IR[15-0])

   Τα σήματα ελέγχου εδώ είναι

   ALUSrcA = 1
   ALUSrcB = 10
   ALUOp = 00

   Β) Αριθμητική – Λογική οδηγία:
   ALUOut = A op B, με τα σήματα ελέγχου
   ALUSrcA = 1
   ALUSrcB = 00
   ALUOp = 10

   Γ) Διακλάδωση:
   if (A == B) PC = ALUOut
   Τα σήματα ελέγχου εδώ είναι

   ALUSrcA = 1
   ALUSrcB = 00
   ALUOp = 01
   PCWriteCond = 1
   PCSource = 01

   Δ) Άλμα:

   PC = PC[31-28] & (IR[25-0] << 2), με σήματα ελέγχου

   PCWrite = 1

4. Προσπέλαση μνήμης ή ολοκλήρωση εντολής τύπου R
   Εδώ έχουμε φόρτωση – αποθήκευση στη μνήμη ή εγγραφή αποτελεσμάτων μιας αριθμητικής – λογικής εντολής.

   Α) Αναφορά στη μνήμη:
   MDR = Memory [ALUOut]
   ή
   Memory [ALUOut] = B
   Τα σήματα ελέγχου παίρνουν τις τιμές

   MemRead = 1 ή MemWrite = 1
   IorD = 1

   Β) Αριθμητική – Λογική οδηγία:
   Reg[IR[15-11]] = ALUOut, με σήματα ελέγχου

   RegDst = 1
   RegWrite = 1
   MemtoReg = 0

5. Ολοκλήρωση ανάγνωσης μνήμης
   Η εντολή φόρτωσης από μνήμη ολοκληρώνεται με εγγραφή του περιεχομένου της.

   Reg[IR[20-16]] = MDR, με σήματα ελέγχου

   MemtoReg = 1
   RegWrite = 1
   RegDst = 0

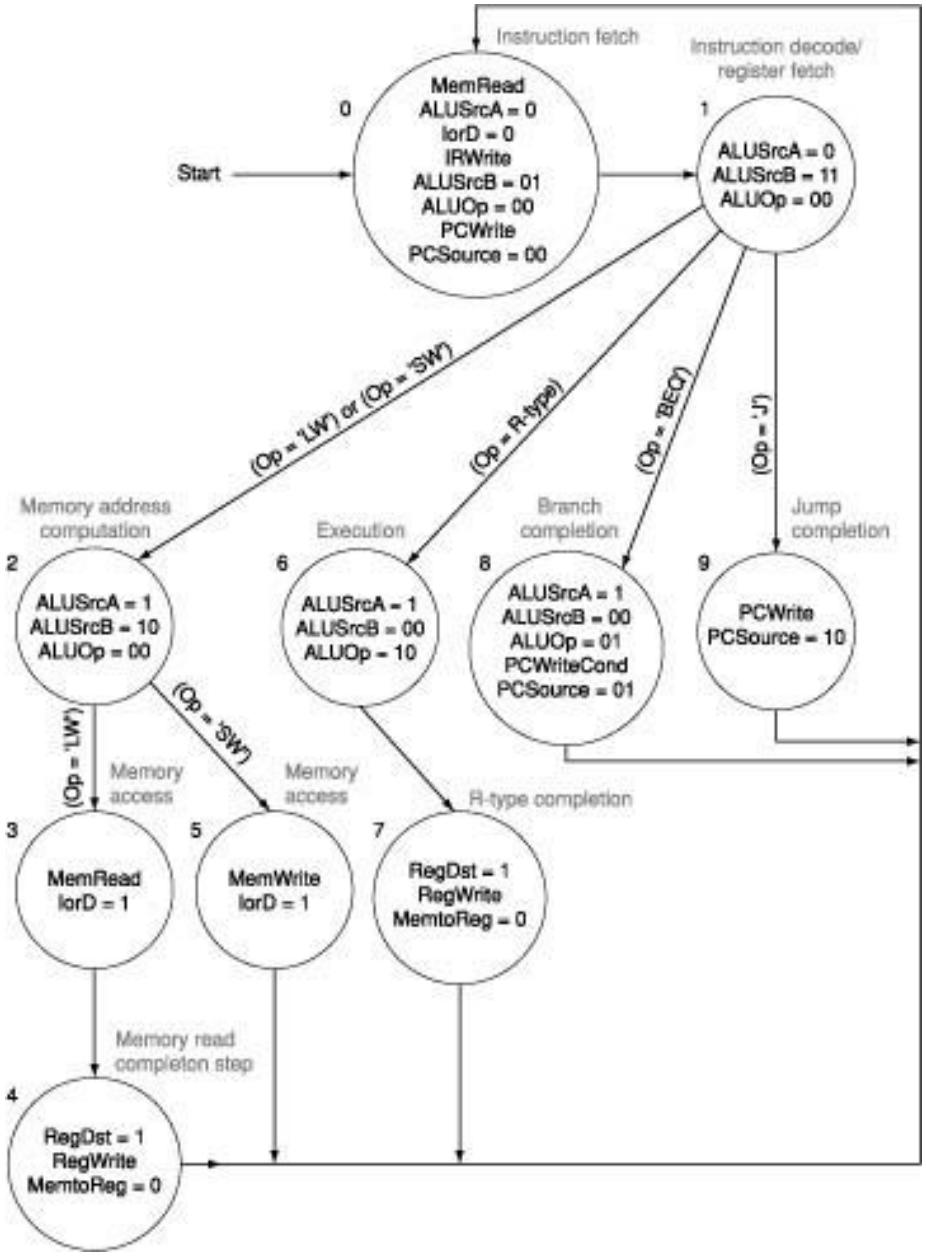
   Τα πέντε παραπάνω βήματα συνοψίζονται στο πίνακα της εικόνας 2.20

Βήμα	Ενέργεια για εντολές τύπου R	Ενέργεια για οδηγίες αναφοράς μνήμης	Ενέργεια για διακλαδώσεις	Ενέργεια για άλματα
1	IR = Memory[PC] PC = PC + 4			
2	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (sign-extend(IR[15-0] << 2)			
3	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A == B) then PC = ALUOut	PC = PC[31-28] & (IR[25-0] << 2)
4	Reg[IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] ή Store: Memory[ALUOut] = B		
5		Load: Reg[IR[20-16]] = MDR		

Πίνακας 2.6 Περίληψη των βημάτων πολλαπλού κύκλου

2.3.3 Καθορίζοντας τον έλεγχο από μία μηχανή πεπερασμένων καταστάσεων

Στην υλοποίηση μονού βήματος(single step) ο έλεγχος καθοριζόταν από απλούς πίνακες αληθείας όπου τα σήματα ελέγχου εξαρτώνταν από την εντολή. Αυτό δεν συμβαίνει για ροή δεδομένων πολλαπλών κύκλων (multicycle). Ο έλεγχος εδώ είναι πιο περίπλοκος δεδομένου του ότι χρειάζεται να καθοριστούν τα κατάλληλα σήματα ελέγχου κάθε βήματος καθώς και το επόμενο, στη σειρά, βήμα. Για αυτό το λόγο χρησιμοποιείται μια μηχανή πεπερασμένων καταστάσεων, όπως φαίνεται στη παρακάτω εικόνα.



Εικόνα 2.12 Ολοκληρωμένη μηχανή ελέγχου πεπερασμένων καταστάσεων

Όλα τα μη χρησιμοποιούμενα σήματα πρέπει να απενεργοποιούνται ή να κρατούν την ίδια τιμή κατά τιε επόμενες καταστάσεις, μέχρι να χρησιμοποιηθούν πάλι.

Όλες οι ρυθμίσεις για τα σήματα ελέγχου σε κάθε κατάσταση φαίνονται στον επόμενο πίνακα.

Σήμα	**Κατάσταση**									
	0	1	2	3	4	5	6	7	8	9
RegDst	0	0	0	0	0	0	0	1	0	0
RegWrite	0	0	0	0	1	0	0	1	0	0
ALUSrcA	0	0	1	1	1	1	1	1	1	0
MemRead	1	0	0	1	0	0	0	0	0	0
MemWrite	0	0	0	0	0	1	0	0	0	0
MemtoReg	0	0	0	0	1	0	0	0	0	0
IorD	0	0	0	1	0	1	0	0	0	0
IRWrite	1	0	0	0	0	0	0	0	0	0
PCWrite	1	0	0	0	0	0	0	0	0	1
PCWriteCond	0	0	0	0	0	0	0	0	1	0
ALUOp	00	00	00	00	00	00	10	10	01	00
ALUSrcB	01	11	10	10	10	10	00	00	00	11
PCSource	00	00	00	00	00	00	00	00	01	10

Πίνακας 2.7 Ρυθμίσεις των σημάτων ελέγχου

3. ΚΑΘΟΡΙΣΜΟΣ ΜΟΝΑΔΩΝ

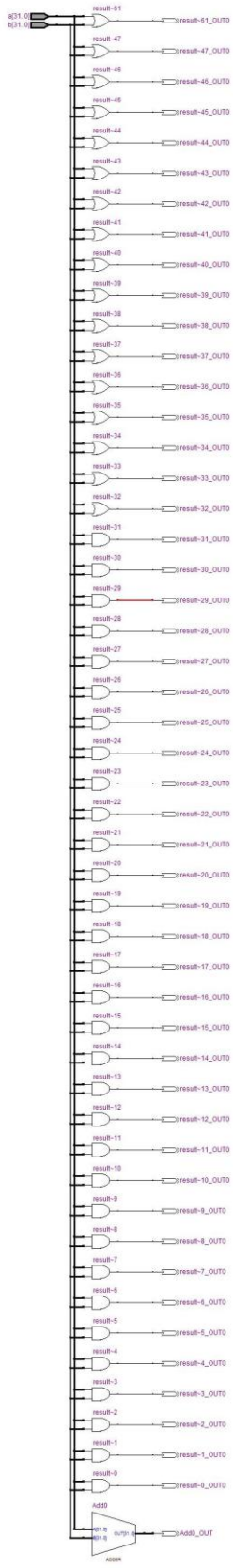
3.1 ALU

3.1.1 Λειτουργική περιγραφή

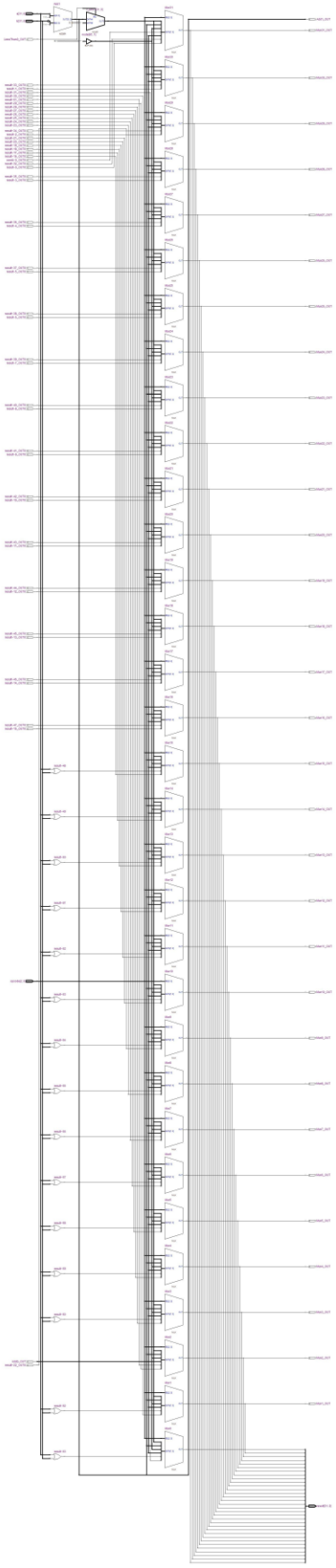
Η μονάδα ALU εκτελεί βασικές αριθμητικές και λογικές πράξεις ελεγχόμενες από τον κώδικα. Το αποτέλεσμα της λειτουργίας γράφεται στην έξοδο. Εάν το αποτέλεσμα είναι μηδέν, ένα επιπλέον μηδενικό bit σηματοδοτεί μία high έξοδο.

Προς το παρόν στις εισόδους μπορούν να εφαρμοστούν οι πράξεις πρόσθεσης (add), αφαίρεσης (sub), λογικού ΚΑΙ (and), Ή (or) και αριστερής ολίσθησης (slt). Οι είσοδοι είναι εύρους 32-bits μη προσημασμένα. Η ανίχνευση υπερχείλισης ή κρατουμένου δεν υποστηρίζεται ακόμα.

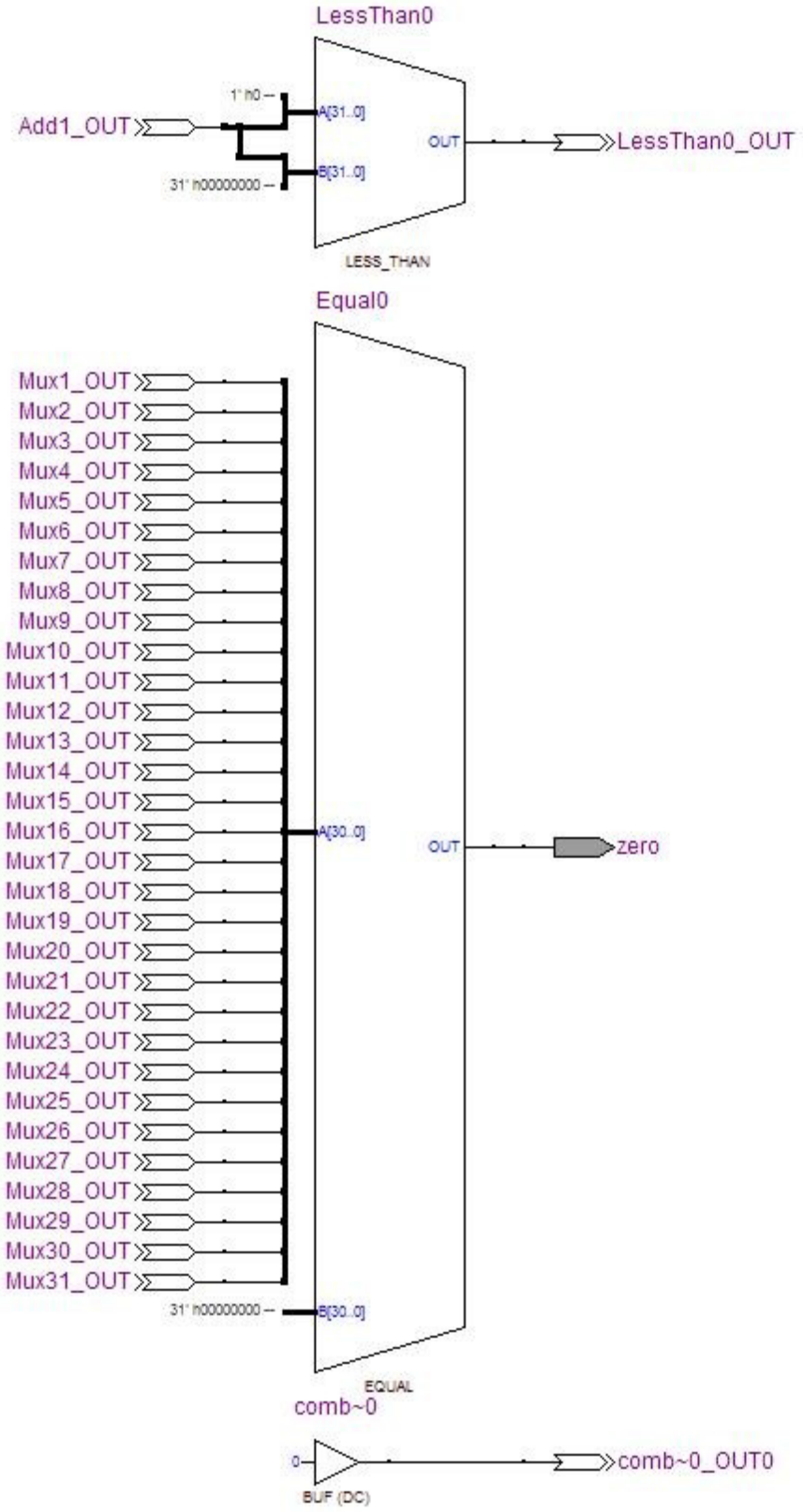
3.1.2 Μπλοκ διάγραμμα



Εικόνα 3.1 ALU 1/3

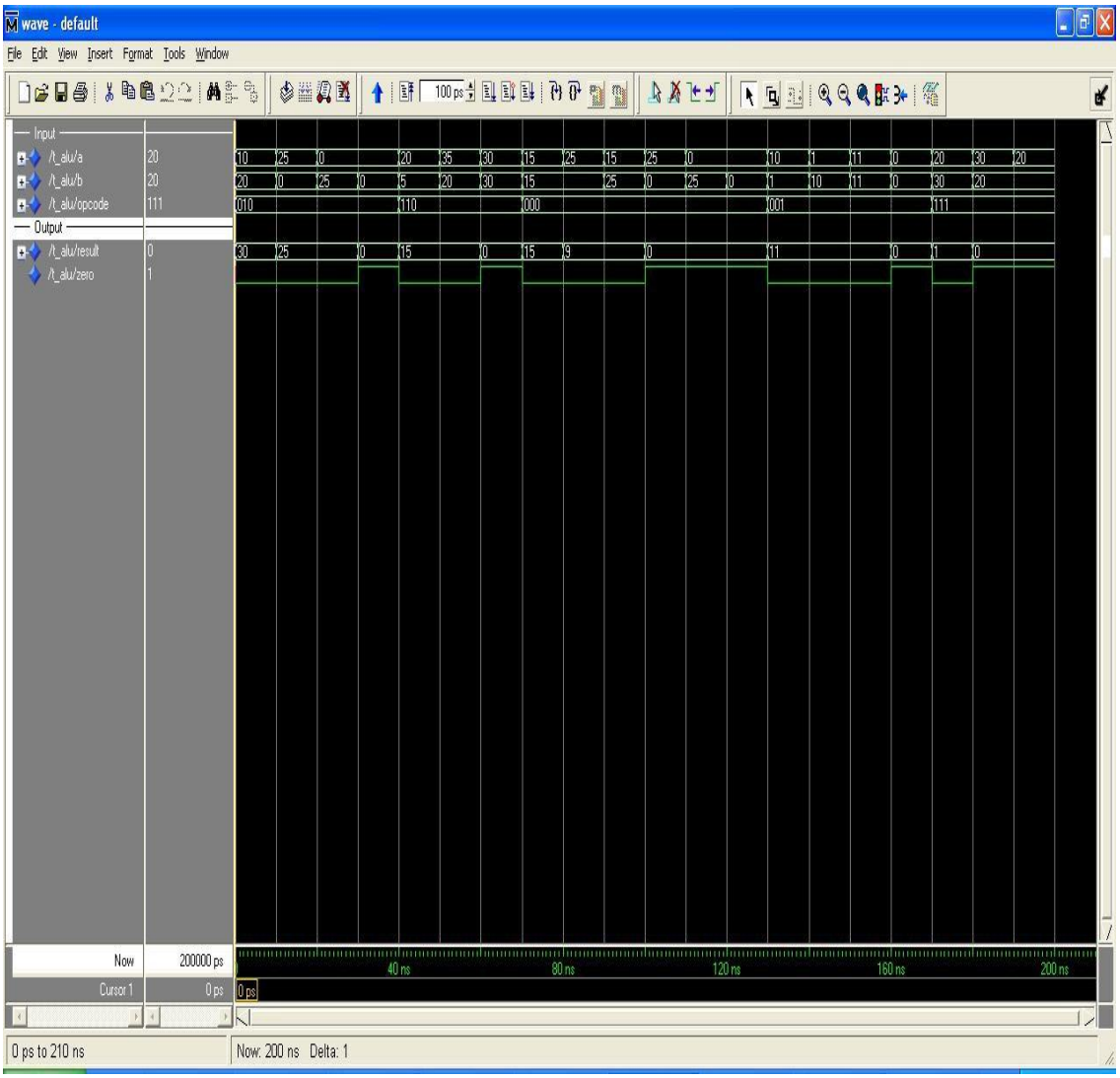


Εικόνα 3.2 ALU 2/3



Εικόνα 3.3 ALU 3/3

3.1.3 Αποτελέσματα προσομοίωσης



Εικόνα 3.4 Αποτελέσματα προσομοίωσης της ALU

3.2 ΜΝΗΜΗ

3.2.1 Περιγραφή λειτουργίας

Τα δεδομένα γράφονται σε ή διαβάζονται από τη μνήμη με data bus των 32-bits. Η μνήμη αποτελείται από τέσσερα μπλοκ RAM με 8-bits εύρος δεδομένων το κάθε ένα.

Ένα σήμα ελέγχου ενεργοποιεί τη μνήμη για εγγραφή δεδομένων ή αλλιώς για ανάγνωση αυτών. Για λόγους αποθήκευσης μια λέξη δεδομένων υποδιαιρείται σε τέσσερα bytes τα οποία εγγράφονται ξεχωριστά στα ram blocks. Αντιστρόφως τα υποδιαιρεμένα bytes συνενώνονται για να μας δώσουν πάλι την αρχική λέξη.

Στο συγκεκριμένο σημείο είναι δυνατή μόνο η ανάγνωση και η εγγραφή λέξεων δεδομένων. Η διευθυνσιοδότηση μισών λέξεων ή μονών bytes δεν επιτρέπεται. Για να πετύχουμε εγγραφή ή ανάγνωση λέξεων πρέπει όλα τα ram blocks να είναι επιλεγμένα. Ως εκ τούτου τα δύο λιγότερο σημαντικά bits δεν ελέγχονται για chip-select λογική.

Τα δεδομένα διευθυνσιοδοτούνται από τον MIPS επεξεργαστή σε ένα εύρος των 32-bits όταν το εύρος διευθύνσεων ενός ram block είναι 8-bits το κάθε ένα. Όλα τα ram blocks είναι συνδεδεμένα στην ίδια διεύθυνση ονομαστικά από mem_address(9 downto 2). Από τη στιγμή που δεν χρησιμοποιούμε όλο το εύρος διευθύνσεων για διευθυνσιοδότηση και chip-selects, οι λέξεις δεδομένων διευθυνσιοδοτούνται από πολλαπλές διευθύνσεις.

Η μνήμη μπορεί να λειτουργήσει σε δύο καταστάσεις:

α) flow - through mode → registered inputs, unregistered outputs

β) pipeline mode → registered inputs, registered outputs

Γενικότερα όταν έχουμε ένα σχέδιο, μπορεί να υπάρχουν σε αυτό καταχωρητές και στην είσοδο και στην έξοδο ώστε να οδηγούν τα δεδομένα που διέρχονται (registered/unregistered I/Os). Μεταξύ αυτών έχουμε τη λογική του σχεδίου μαζί με, ενδεχομένως, κάποια ακόμα στάδια καταχωρητών (π.χ. για pipelining ή διάφορες άλλες εσωτερικές καταστάσεις).

Το εργαλείο σύνθεσης μπορεί έτσι να ασχοληθεί με τη βέλτιστη λύση ανάλογα και με τους περιορισμούς του ρολογιού. Χωρίς τους καταχωρητές εισόδου η καθυστέρηση (delay) δε θα ήταν υπολογίσιμη.

Τώρα αν έχουμε πολλές μονάδες με τους αντίστοιχους καταχωρητές εισόδου/εξόδου των, μπορούμε να τις συνδυάσουμε μεταξύ τους και να είμαστε αρκετά σίγουροι ότι θα δουλέψουν στα σωστά χρονικά περιθώρια αν

αυτά είναι κάτω των χρονικών περιορισμών του συστήματος. Οι καταχωρητές εξόδου της μίας μονάδας συνδέονται με την είσοδο της επόμενης κ.ο.κ. Με τη λογική αυτή έχουμε τα πλεονεκτήματα ότι:

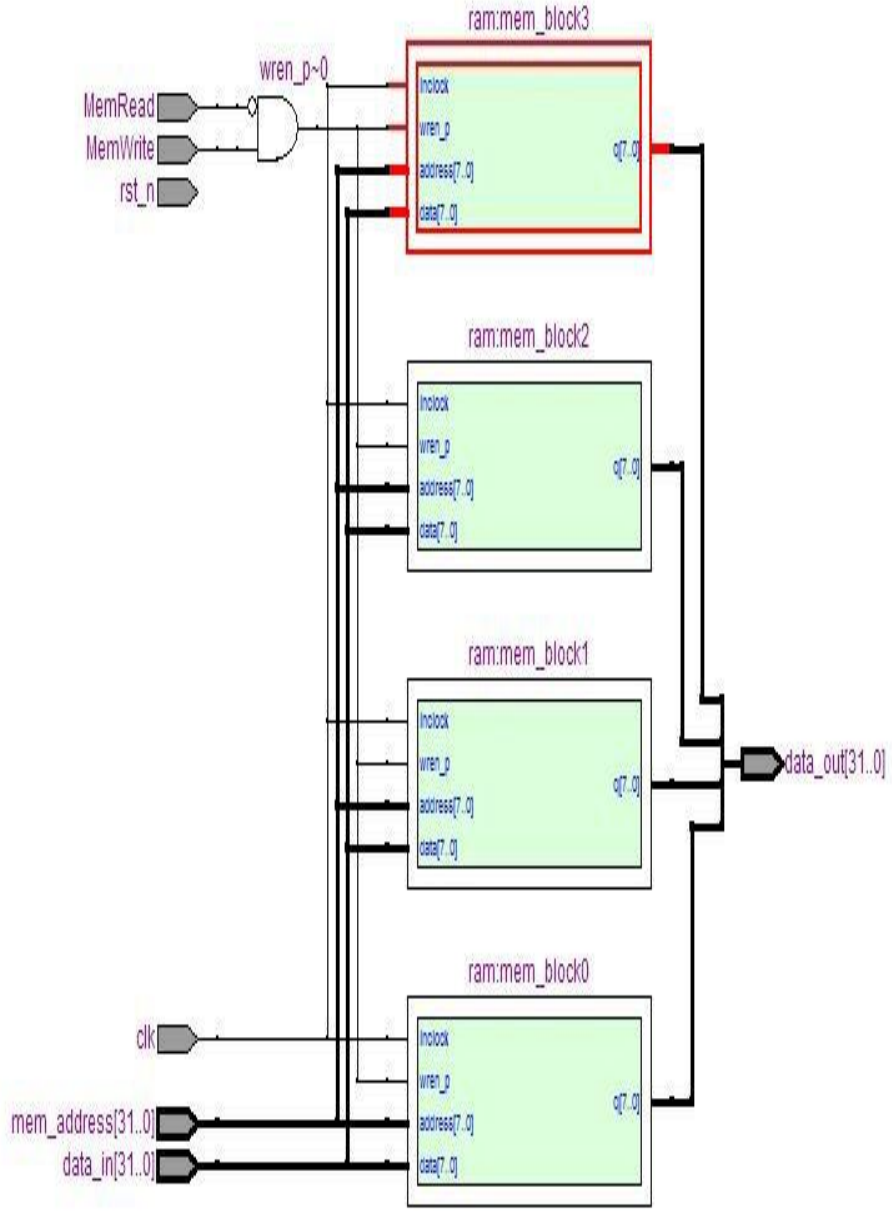
α) Τίποτα δε χρειάζεται να αλλάξει. Όλες οι μονάδες λειτουργούν κανονικά.

β) Η δρομολόγηση γίνεται πιο απλή λόγω των διπλών επιπέδων καταχωρητών.

γ) Βελτιώνεται η αξιοποίηση του χρόνου απόκρισης.

Αν ο synthesizer του συστήματος δεν υποστηρίζει αρχεία αρχικοποίησης μνήμης, τότε δεν είναι δυνατή η απόδοση του συστήματος σε πραγματικό υλικό (hardware).

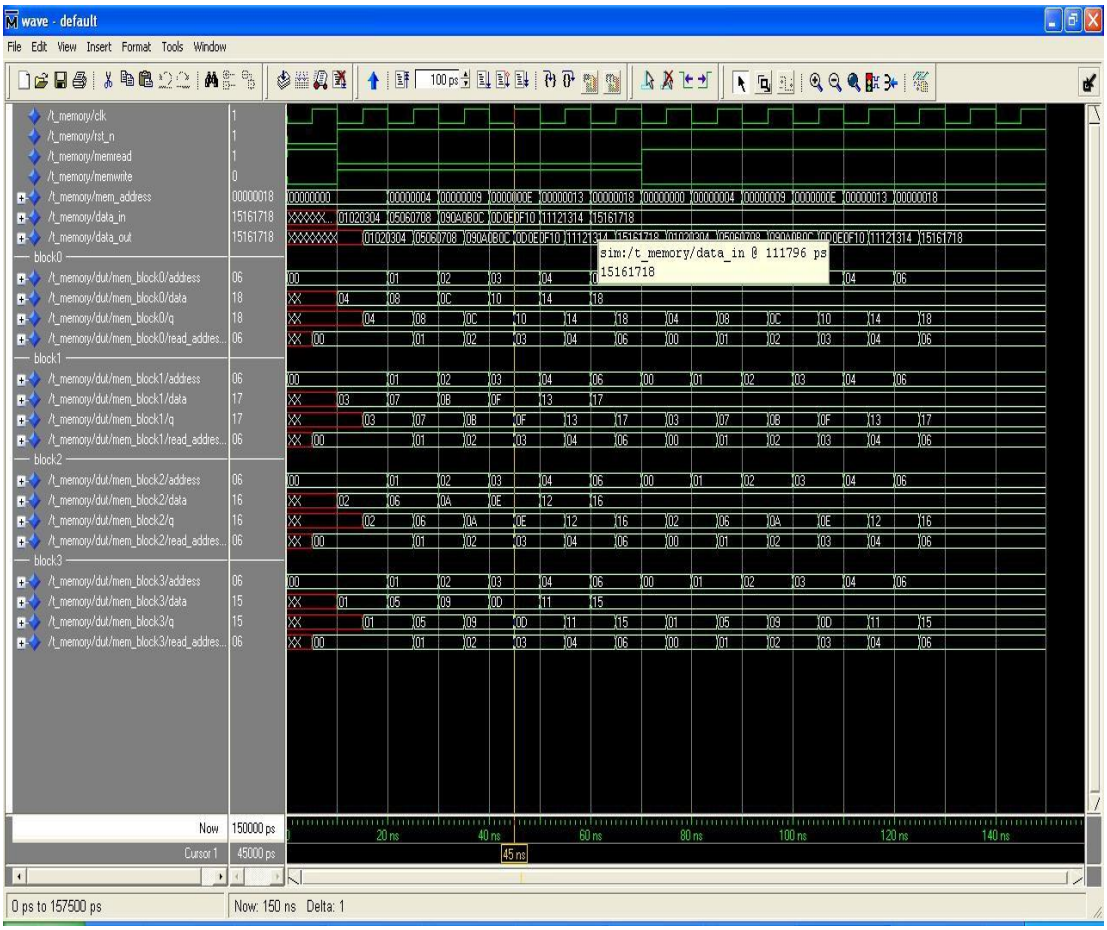
3.2.2 Μπλοκ διάγραμμα



Εικόνα 3.5 Μνήμη

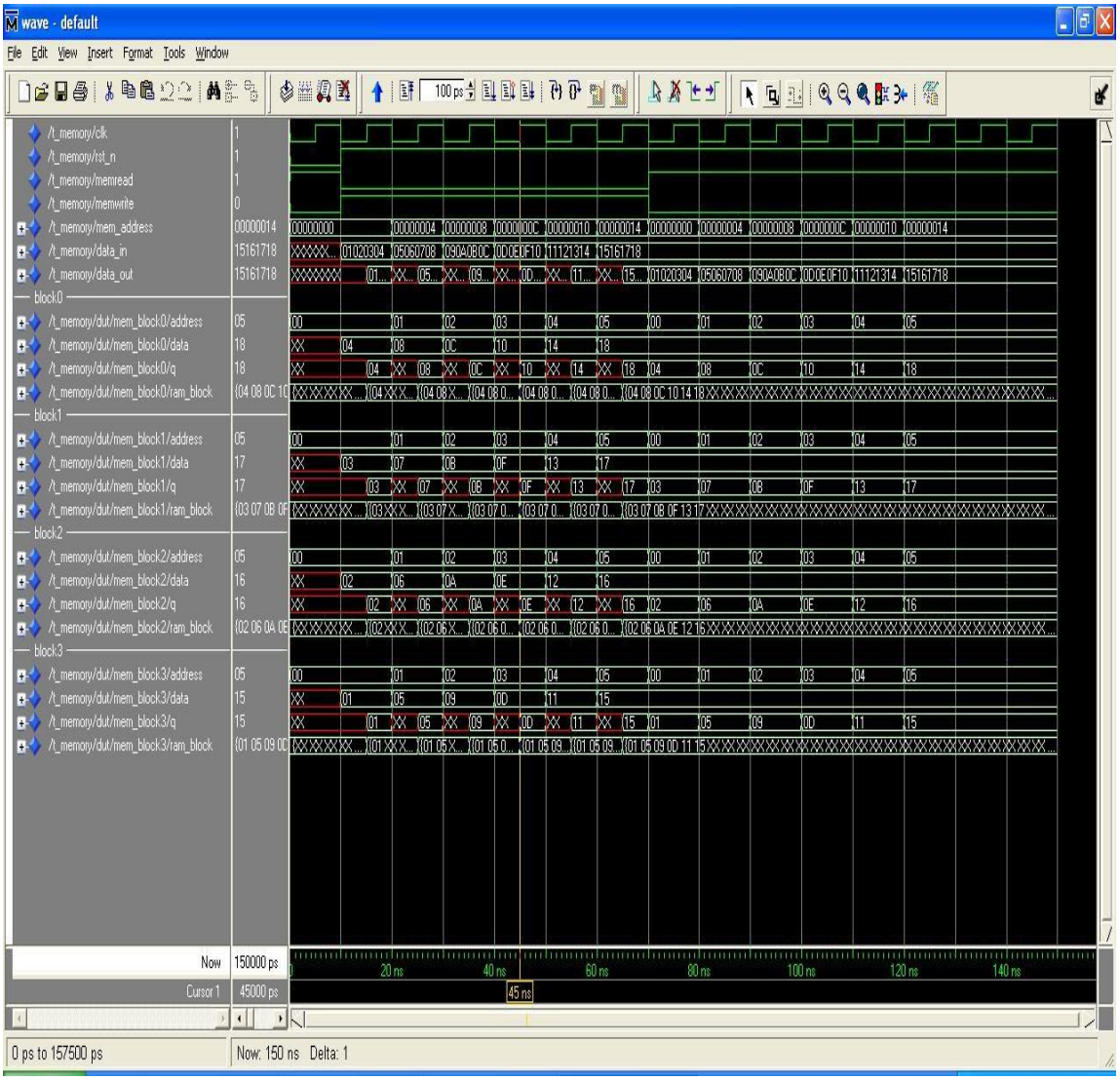
3.2.3 Αποτελέσματα προσομοίωσης

Η εικόνα 4.6 δείχνει τα αποτελέσματα της προσομοίωσης με καταχωρημένα δεδομένα εξόδου.



Εικόνα 3.6 Αποτελέσματα προσομοίωσης της μνήμης (registered outputs)

Η εικόνα 3.7 δείχνει τα αποτελέσματα της προσομοίωσης με μη καταχωρημένη έξοδο. Παρατηρούμε ότι η προσομοίωση περιέχει άγνωστες τιμές λόγω του ότι δεν υποστηρίζονται αρχεία αρχικοποίησης της μνήμης.



Εικόνα 3.7 Αποτελέσματα προσομοίωσης μνήμης (unregistered outputs)

3.3 ΄ΕΛΕΓΧΟΣ

3.3.1 Περιγραφή λειτουργίας

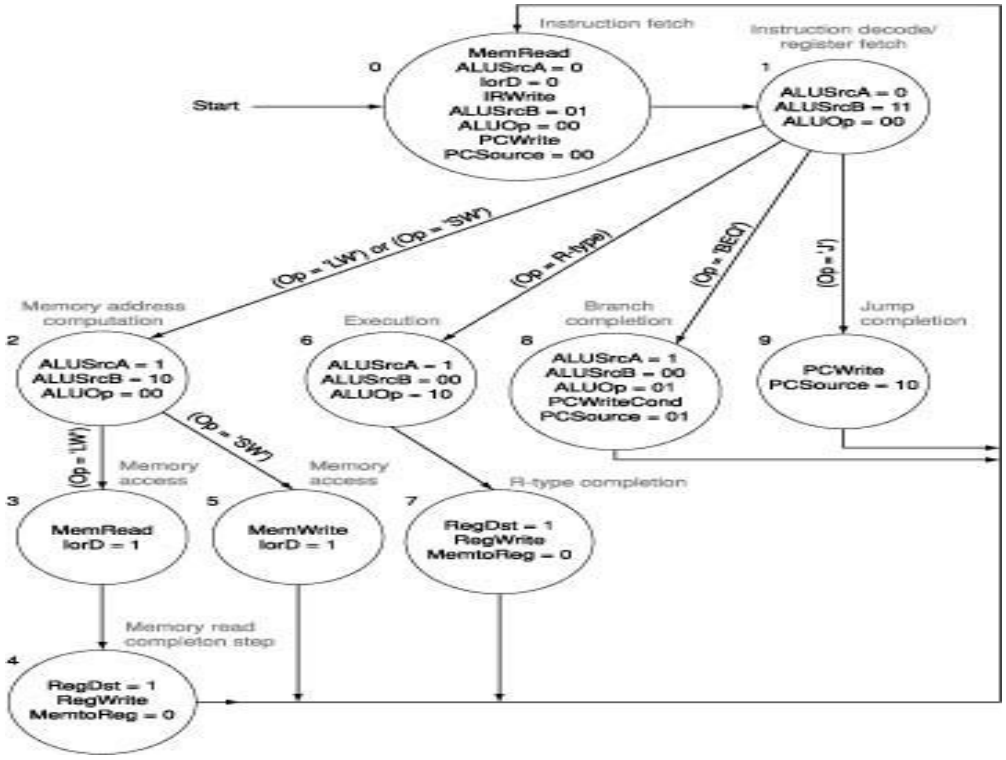
Ο έλεγχος του επεξεργαστή γίνεται κατανοητός μέσω μιας Μηχανής Πεπερασμένων Καταστάσεων (FSM), όπως περιγράφεται στη παράγραφο 2.3.3.

Ως είσοδο για τη Μηχανή Καταστάσεων έχουμε τα 6 περισσότερο σημαντικά bits από το πεδίο λειτουργίας (function field) που περιέχει την εντολή.

Οι έξοδοι της Μηχανής είναι τα σήματα ελέγχου των ενιαίων λειτουργικών μονάδων του επεξεργαστή ειδικά των πολυπλεκτών δεδομένων.

Ο Κώδικας Λειτουργίας (Operation Code) της ALU είναι καταχωρημένος σε ένα πίνακα αληθείας και ο αντίστοιχος κώδικας πράξης (Opcode) παράγεται σε σχέση με το σήμα ALUOp της Μηχανής Καταστάσεων και τα 6 λιγότερο σημαντικά bits του πεδίου λειτουργίας που περιέχει τη πληροφορία εάν θα γίνει χρήση της αριθμητικής οδηγίας-εντολής ή της λογικής.

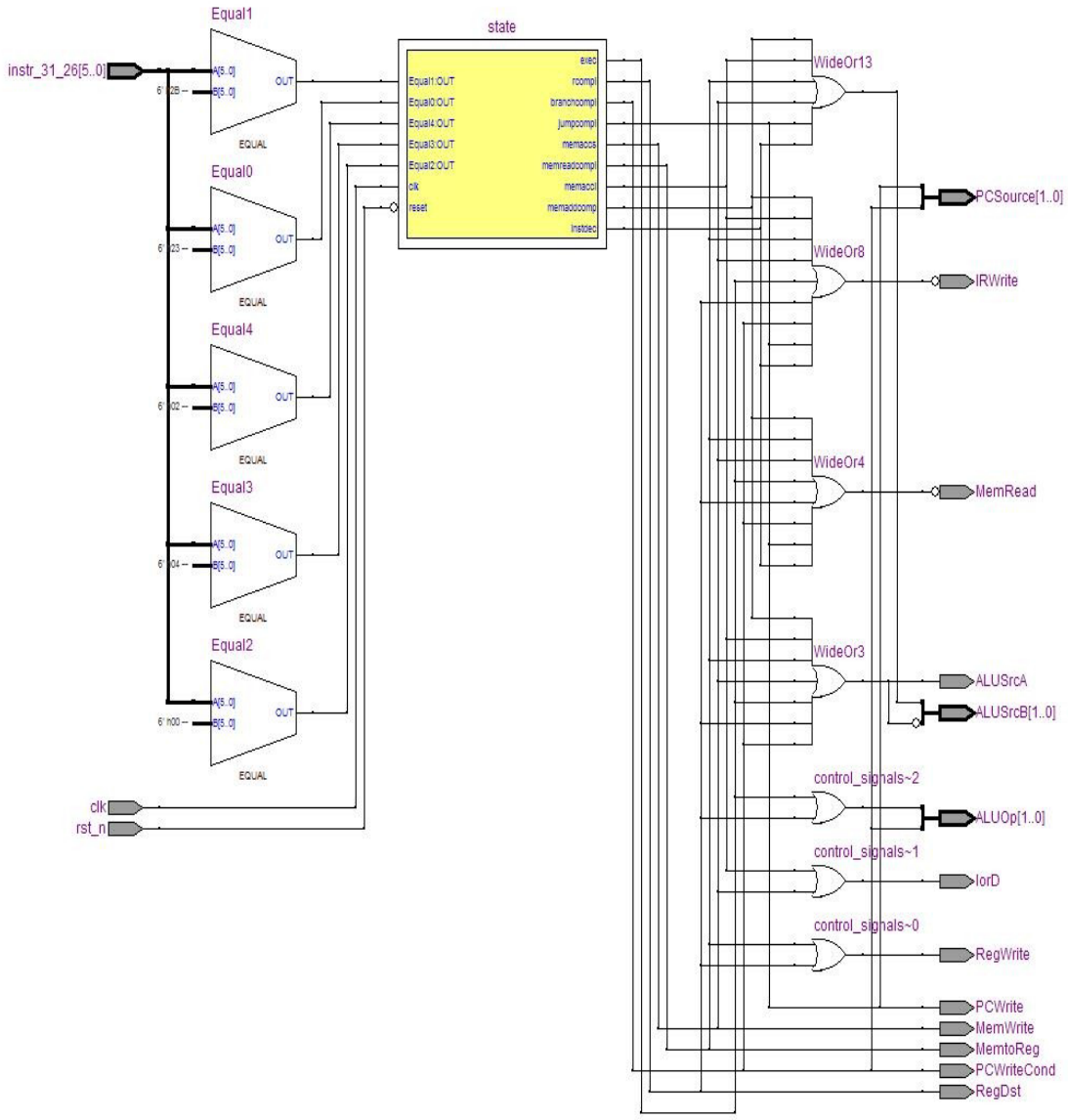
3.3.2 Διάγραμμα καταστάσεων



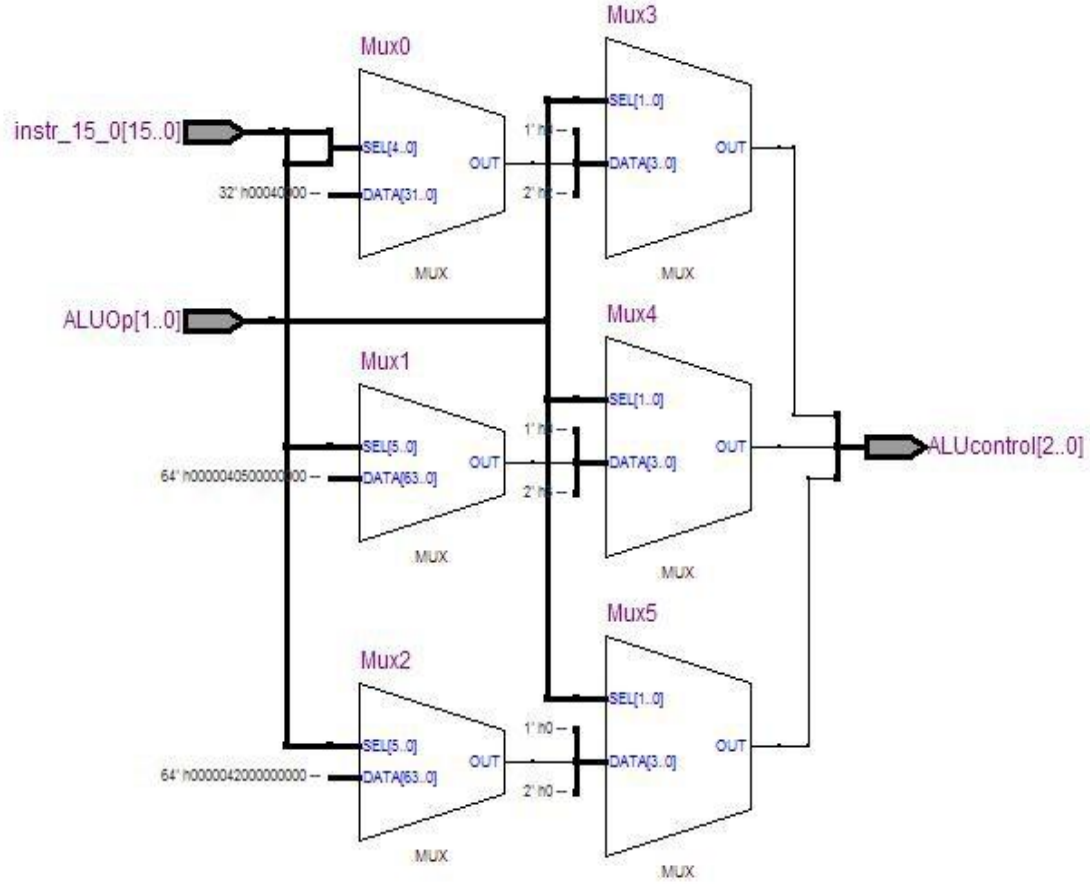
Εικόνα 3.8 Έλεγχος FSM

Έχει προστεθεί και μια επιπλέον κατάσταση σφάλματος η οποία οδηγεί σε αδιέξοδο. Ενεργοποιείται σε περίπτωση εμφάνισης μιας άγνωστης εντολής.

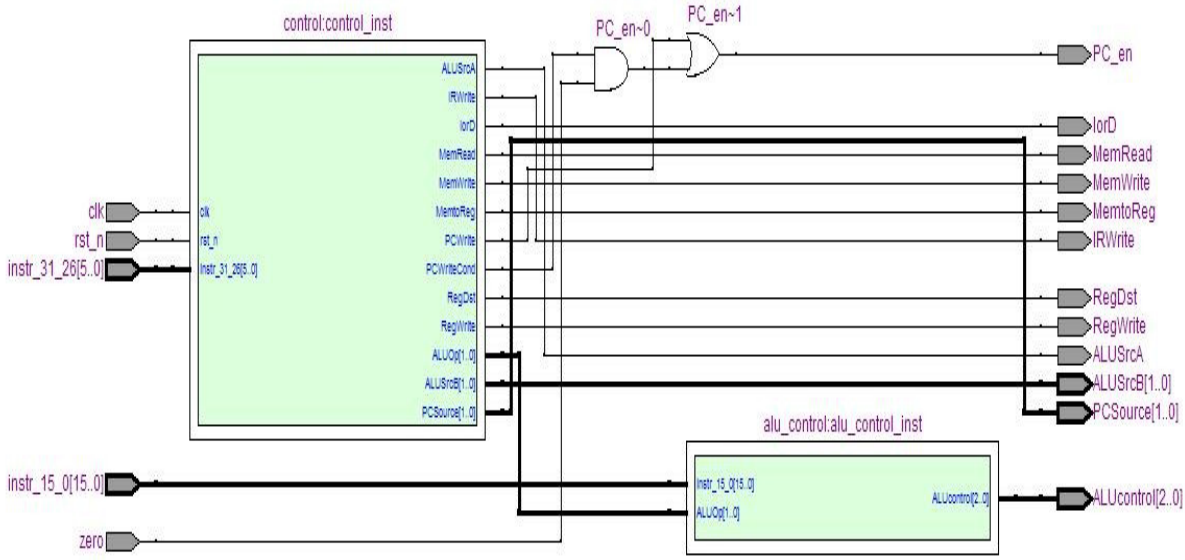
3.3.3 Μπλοκ διάγραμμα



Εικόνα 3.9 Έλεγχος FSM

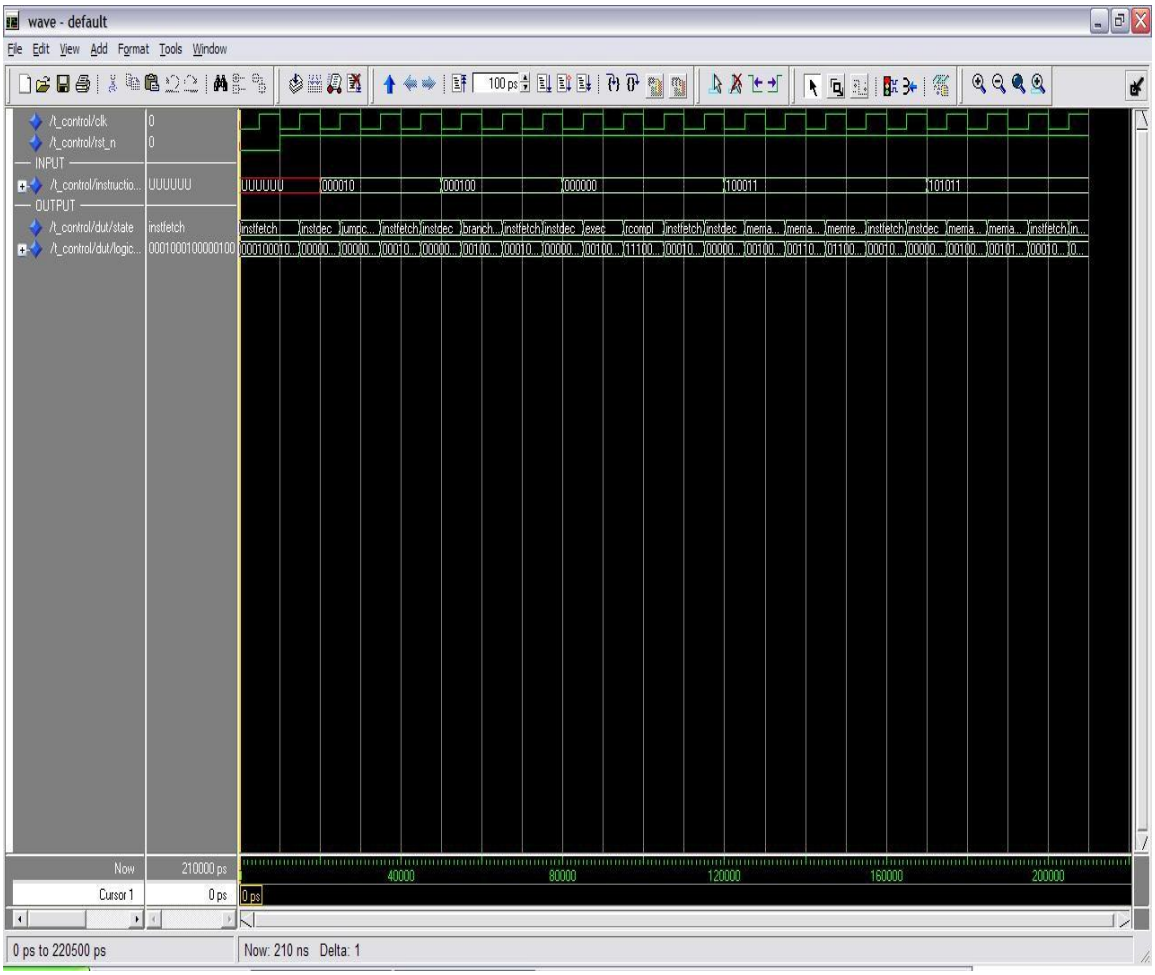


Εικόνα 3.10 Έλεγχος ALU



Εικόνα 3.11 Έλεγχος

3.3.4 Αποτελέσματα προσομοίωσης



Εικόνα 3.12 Αποτελέσματα προσομοίωσης ελέγχου FSM

3.4 ΡΟΗ ΔΕΔΟΜΕΝΩΝ (DATA PATH)

Η επικάλυψη εντολών στην αρχιτεκτονική υπολογιστών είναι μια τεχνική που εκφράζεται μέσω της μικροαρχιτεκτονικής του συστήματος και έχει ως σκοπό τη βελτίωση του IPC (instructions per cycle) και της γενικότερης απόδοσης του συστήματος. Η λογική πίσω από τη τεχνική αυτή είναι ότι η κάθε εντολή του επεξεργαστή κατακερματίζεται σε απλές μικρές λειτουργίες οι οποίες αν ενωθούν παράγουν το τελικό αποτέλεσμα. Λόγω μιας ανεξαρτησίας των λειτουργιών αυτών, είναι δυνατή η εκτέλεση της κάθε μίας από αυτές σε διαφορετικό κύκλο μηχανής. Για την περεταίρω αξιοποίηση του συστήματος η κάθε εντολή ξεκινάει την εκτέλεσή της πριν ολοκληρωθεί η εκτέλεση της προηγούμενης. Η μερική αυτή επικάλυψη της εκτέλεσης των εντολών είναι μια τεχνική που χρησιμοποιείται ευρύτατα και σε σύγχρονα συστήματα, ενώ η πρώτη της μορφή παρουσιάστηκε στους επεξεργαστές τύπου RISC. Ο MIPS σαν μονοπύρηνος επεξεργαστής εισήγαγε το μοντέλο αυτό υλοποιώντας τα πέντε στάδια μερικής επικάλυψης γνωστά ως: ανάκληση εντολής (instruction fetch), αποκωδικοποίηση εντολής (instruction decode), εκτέλεση εντολής (execution), προσπέλαση μνήμης (memory access) και εγγραφή αποτελεσμάτων (memory write back). Η κάθε εντολή της αρχιτεκτονικής αυτής περνάει από όλα τα στάδια εκτελώντας τις αντίστοιχες λειτουργίες του κάθε σταδίου σε ένα κύκλο μηχανής. Παρακάτω εξηγείται η λειτουργικότητα του κάθε σταδίου και ο αντίκτυπός του στη συνολική λειτουργία του επεξεργαστή.

Τα παραπάνω τμήματα συντίθενται από μόνα τους και μετά συνδυάζονται με το μονοπάτι δεδομένων.

3.4.1 Ανάκληση εντολής - Instruction Fetch

3.4.1.1 Περιγραφή λειτουργίας

Το Instruction Fetch Block περιέχει τον PC (Program Counter) και τους καταχωρητές Instruction Reg και Memory Data Reg.

Το τμήμα αυτό παρέχει στα δεδομένα και τις οδηγίες-εντολές στην μνήμη.

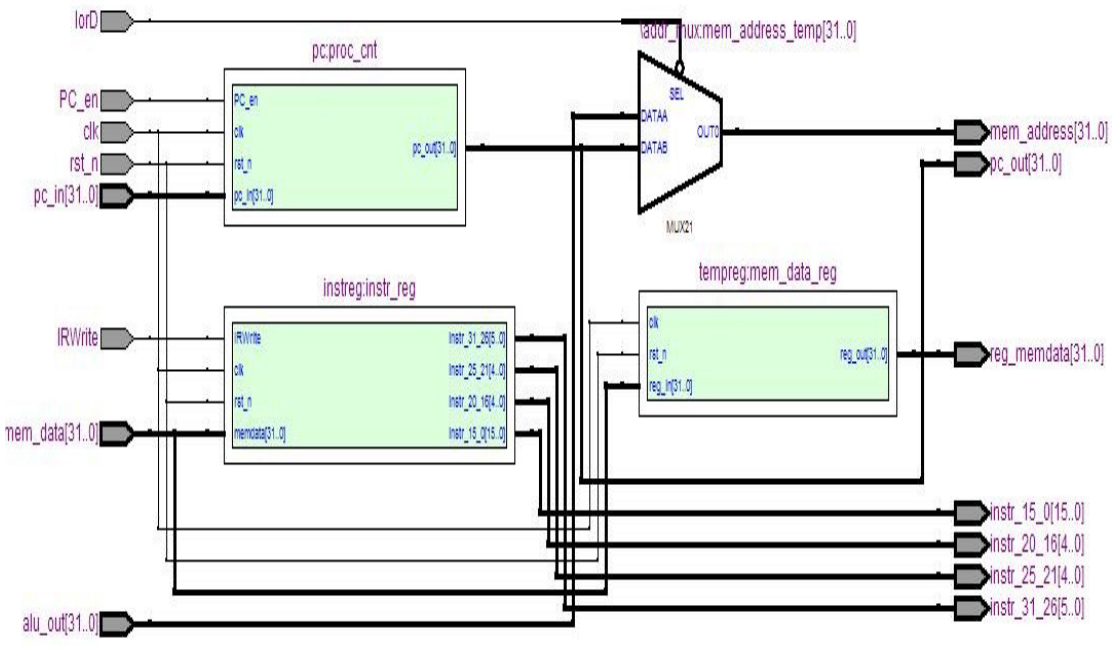
Καθώς στο στάδιο αυτό γίνεται πρόσβαση στη κρυφή μνήμη (cache) θα είναι καλό να αναφερθεί ο διαχωρισμός μνήμης που ορίζεται στον MIPS. Η κρυφή μνήμη του επεξεργαστή διαχωρίζεται σε δύο διακριτά τμήματα: τη μνήμη εντολών (instruction cache) και τη μνήμη δεδομένων (data cache). Στη πρώτη αποθηκεύονται μόνο οι εντολές προς εκτέλεση στον επεξεργαστή ενώ στη δεύτερη μόνο τα δεδομένα τα οποία προσπελαύνονται κατά τη διάρκεια

της εκτέλεσης του προγράμματος. Τέτοια δεδομένα μπορεί να είναι δομές όπως πίνακες, λίστες, κλάσεις και δέντρα.

Κατά την ανάκληση εντολής γίνεται πρόσβαση στη κρυφή μνήμη εντολών, στη διεύθυνση που δίνει ο ειδικός καταχωρητής pc (program counter) ώστε να ανακληθεί η επόμενη εντολή προς εκτέλεση. Μετά την ανάκληση ο καταχωρητής αυξάνεται ώστε να δείχνει στην επόμενη θέση μνήμης για την ανάκληση της εντολής που ακολουθεί.

Στο τέλος του κύκλου μηχανής η εντολή αποθηκεύεται στο καταχωρητή επικάλυψης (IF/ID), όπως βλέπουμε στην εικόνα 1.2, ο οποίος τη προωθεί στο επόμενο στάδιο, στην αρχή του επόμενου κύκλου.

3.4.1.2 Μπλοκ διάγραμμα



Εικόνα 3.13 Instruction Fetch

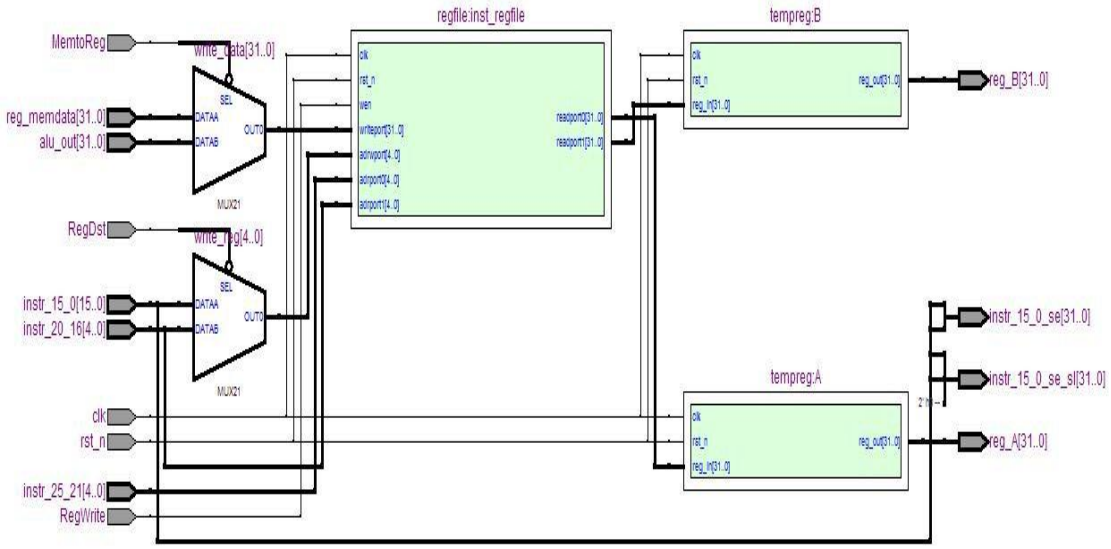
3.4.2 Αποκωδικοποίηση εντολής - Instruction Decode

3.4.2.1 Περιγραφή λειτουργίας

Το Instruction Decode Block γράφει την οδηγία από τον Instruction Register στο αρχείο καταχωρητών (Register File) και υπολογίζει το δεύτερο τελεστή για μια οδηγία διακλάδωσης (Branch Instruction) είτε για μια οδηγία sw- ή lw-.

Στο κύκλο αυτό γίνεται η αποκωδικοποίηση της εντολής από το προηγούμενο στάδιο. Η εντολή διασπάται από ένα κύκλωμα ελέγχου στα βασικά της πέντε πεδία: το κωδικό λειτουργίας (operation code) που καθορίζει τη πράξη που θα γίνει στην αριθμητική και λογική μονάδα, του τρεις καταχωρητές (Rs, Rt, Rd) που θα χρησιμοποιηθούν για ανάγνωση και εγγραφή δεδομένων και το τελευταίο πεδίο που είτε περιέχει βοηθητική πληροφορία για τη πράξη που θα γίνει, είτε το immediate μέρος της εντολής, δηλαδή ένα σταθερό αριθμό που θα χρησιμοποιηθεί για να παραχθεί το τελικό αποτέλεσμα. Μετά την αποκωδικοποίηση εντολής το κύκλωμα ελέγχου παράγει τα κατάλληλα σήματα ανάλογα με την εκάστοτε εντολή, ελέγχοντας τη λειτουργία της αριθμητικής και λογικής μονάδας, όπως και τα σήματα επιλογής για τους πολυπλέκτες που θα χρησιμοποιηθούν. Να σημειωθεί επίσης ότι σε περίπτωση που η εντολή έχει immediate μέρος τότε γίνεται και επέκταση προσήμου καθώς αυτό είναι 16-bits ενώ για μια πράξη στην ALU (αριθμητική και λογική μονάδα) απαιτούνται τελεστέοι των 32-bits. Τέλος γίνεται πρόσβαση στο φάκελο καταχωρητών ώστε να γίνει η ανάκληση των περιεχομένων των καταχωρητών που χρειάζονται για την εκτέλεση της εντολή. Στο τέλος του κύκλου γίνεται αποθήκευση των απαραίτητων δεδομένων στο καταχωρητή επικάλυψης (ID/EX – εικόνα 1.2) για τη συνέχιση εκτέλεσης της εντολής.

3.4.2.2 Μπλοκ διάγραμμα



Εικόνα 3.14 Instruction Decode

3.4.3 Εκτέλεση εντολής - Execution

3.4.3.1 Περιγραφή λειτουργίας

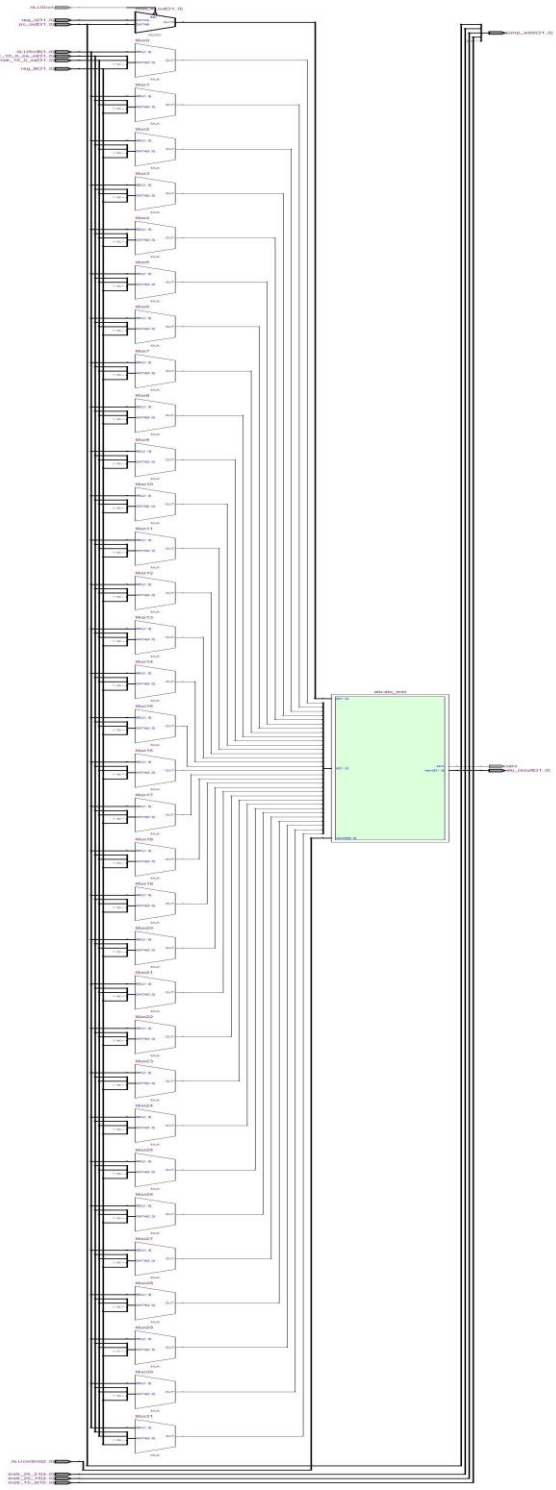
Το τμήμα Execution περιέχει την ALU σαν κύριο συστατικό και υπολογίζει το επιθυμητό αποτέλεσμα μιας εντολής.

Υπολογίζει επίσης και τη διεύθυνση στόχου άλματος (jump target address) την οποία παρέχει στο Memory Writeback Block.

Οι τελεστές που έχουν φορτωθεί στην ALU επιλέγονται από δύο πολυπλέκτες που είναι ευαίσθητοι στα σήματα ALUSrcA και ALUSrcB.

Έχοντας έτοιμα τα κατάλληλα σήματα για την πράξη στην αριθμητική και λογική μονάδα καθώς και τους τελεστέους που είναι απαραίτητοι για την πράξη, στο κύκλο αυτό γίνεται η εκτέλεση της εντολής στο κύκλωμα της ALU. Η εκτέλεση αυτή πρακτικά είναι η διεκπεραίωση της πράξης που ορίσθηκε από το στάδιο της αποκωδικοποίησης εντολής με τελεστέους είτε δύο καταχωρητές, είτε ένα καταχωρητή και το immediate/σταθερό μέρος της εντολή. Η πράξη αυτή ολοκληρώνεται σε ένα κύκλο μηχανής ενώ το αποτέλεσμά της αποθηκεύεται στον EX/MEM καταχωρητή επικάλυψης.

3.4.3.2 Μπλοκ διάγραμμα



Εικόνα 3.15 Execution

3.4.4 Προσπέλαση μνήμης – Memory Access

3.4.4.1 Περιγραφή λειτουργίας

Στο κύκλο αυτό γίνεται προσπέλαση (αν το επιβάλλει η λειτουργία της εντολής) της κρυφής μνήμης δεδομένων του επεξεργαστή. Αυτό συμβαίνει για τη μεταφορά δεδομένων από και προς τη μνήμη. Η μνήμη δέχεται σαν είσοδο μια δυαδική διεύθυνση, δεδομένα εύρους 32-bits και ένα bit επιλογή λειτουργίας (φόρτωση ή αποθήκευση). Στη συνέχεια αποθηκεύει στη διεύθυνση τα δεδομένα που δέχθηκε σαν είσοδο είτε βγάζει σαν έξοδο τα δεδομένα που είναι αποθηκευμένα στην αντίστοιχη διεύθυνση, ανάλογα με τη τιμή του bit επιλογής λειτουργίας. Η έξοδος της μνήμης εφ' όσον υπάρχει αποθηκεύεται στο καταχωρητή επικάλυψης MEM/WB για τη χρήση της στο επόμενο στάδιο επικάλυψης. Αν η έξοδος της μνήμης δεδομένων δεν υπάρχει τότε στον MEM/WB αποθηκεύεται η έξοδος της αριθμητικής και λογικής μονάδας.

3.4.5 Εγγραφή αποτελεσμάτων - Memory Writeback

3.4.5.1 Περιγραφή λειτουργίας

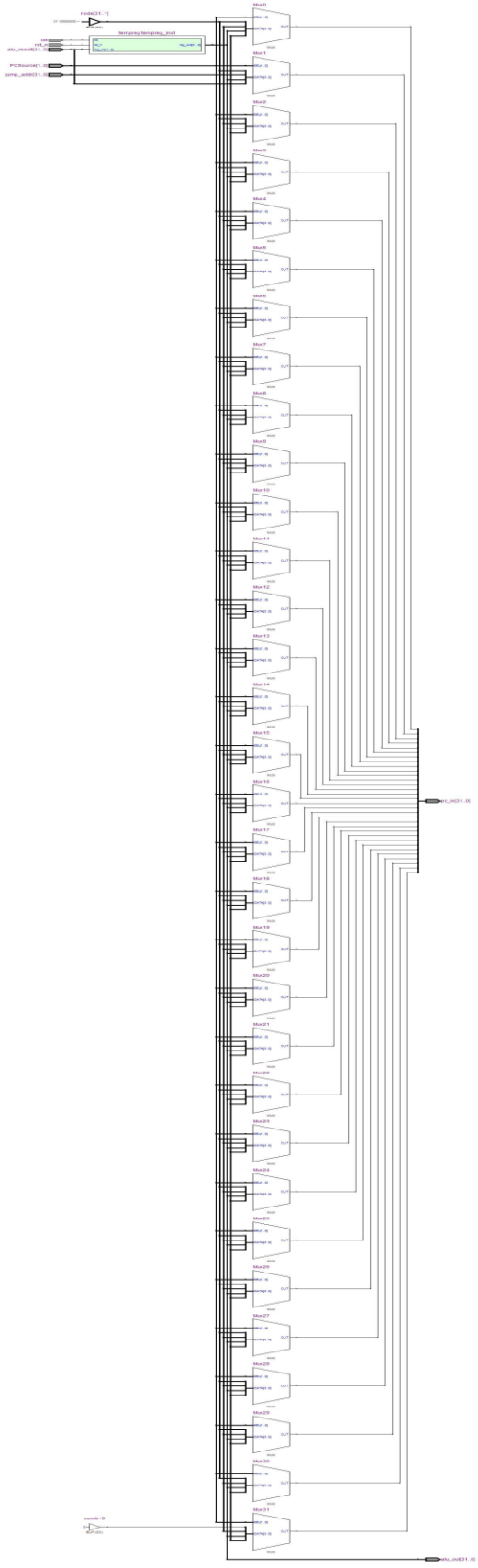
Το Memory Writeback Block αποτελείται από το καταχωρητή ALUOut και ένα πολυπλέκτη με πηγαίο σήμα το PCSource.

Το μπλοκ αυτό οδηγεί το αποτέλεσμα του υπολογισμού είτε πίσω στη μνήμη είτε στο αρχείο καταχωρητών.

Ο πολυπλέκτης επαναφέρει την επόμενη τιμή του PC σε εξάρτηση και με το σήμα PCSource.

Κατά την εγγραφή του αποτελέσματος τα περιεχόμενα του καταχωρητή επικάλυψης MEM/WB αποθηκεύονται στο καταχωρητή προορισμού της εντολής που βρίσκεται στο φάκελο καταχωρητών. Το στάδιο αυτό αποτελεί και το τελευταίο στάδιο εκτέλεσης για μια εντολή στον επεξεργαστή MIPS.

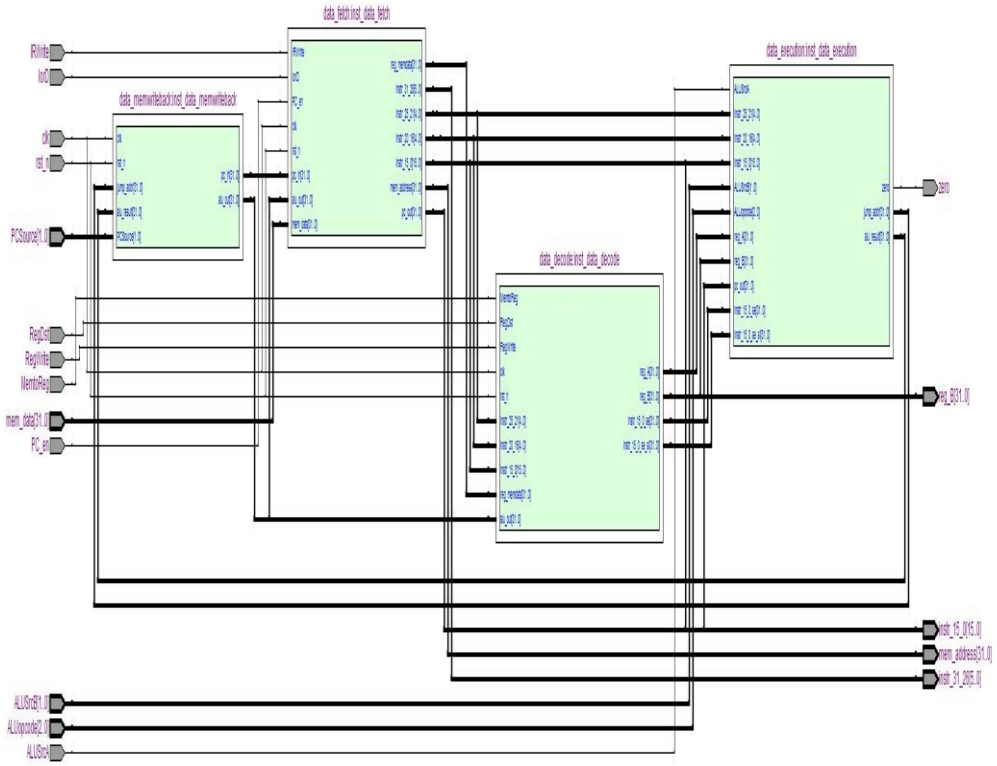
3.4.5.2 Μπλοκ διάγραμμα



Εικόνα 3.16 Memory Writeback

3.4.6 Κανάλι δεδομένων - Data Path

3.4.6.1 Μπλοκ διάγραμμα



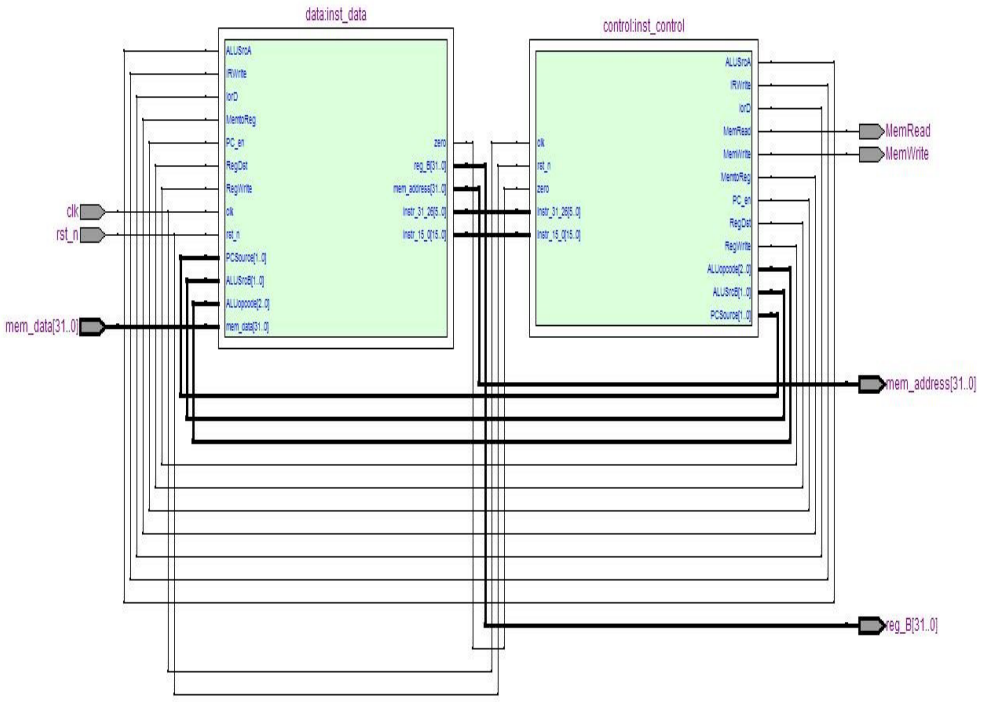
Εικόνα 3.16 Datapath

3.5 ΕΠΕΞΕΡΓΑΣΤΗΣ ΚΑΙ ΜΝΗΜΗ

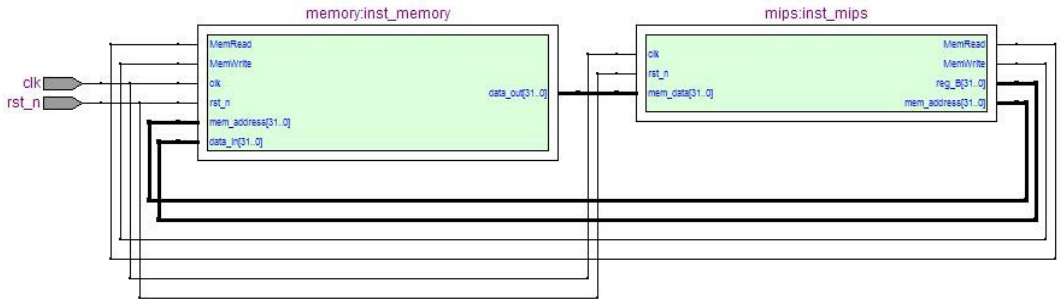
3.5.1 Περιγραφή λειτουργίας

Τα δύο κομμάτια ροής δεδομένων και ελέγχου (Datapath, Controlpath) συνδυάζονται μέσα στη μονάδα επεξεργασίας. Μαζί και με τη μνήμη συνιστούν τον επεξεργαστή.

3.5.2 Μπλοκ διάγραμμα



Εικόνα 3.17 Μονάδα επεξεργασίας (Datapath & Controlpath)



Εικόνα 3.18 Μονάδα επεξεργασίας και Μνήμη

3.6 Το σύνολο εντολών του MIPS

Κατά τη διαδικασία σχεδίασης ενός επεξεργαστή ο ορισμός και η δημιουργία του συνόλου εντολών του είναι μία από τις πρωταρχικές και πιο σημαντικές ενέργειες. Με τον όρο σύνολο εντολών περιγράφουμε όλες τις εντολές που ο εκάστοτε επεξεργαστής είναι ικανός να εκτελέσει. Η σχεδίαση του συνόλου εντολών γίνεται από τον αρχιτέκτονα του συστήματος με επιλογές που καθιστούν τον επεξεργαστή αποδοτικό σε ορισμένες ενέργειες και μη αποδοτικό σε άλλες. Στη περίπτωση του MIPS οι εντολές του είναι των 32-bits και χωρίζονται σε τρεις μεγάλες κατηγορίες: τις εντολές τύπου R, τις εντολές τύπου I και τις εντολές τύπου J. Η κατηγοριοποίηση αυτή προκύπτει όχι από τη λειτουργικότητα των εντολών αλλά από το ελαφρώς διαφοροποιημένο μοτίβο κωδικοποίησής τους. Πιο συγκεκριμένα στις εντολές τύπου R περιλαμβάνονται λογικές και αριθμητικές πράξεις που γίνονται μεταξύ δύο καταχωρητών με το αποτέλεσμα της πράξης αυτής να αποθηκεύεται σε ένα καταχωρητή προορισμού. Οι εντολές τύπου I είναι εντολές που πραγματοποιούνε πράξεις αντίστοιχες με αυτές των R τύπων αλλά οι τελεστέοι είναι ένας καταχωρητής και μία αριθμητική σταθερά, ενώ το αποτέλεσμα αποθηκεύεται στη μνήμη ή σε ένα καταχωρητή προορισμού. Οι εντολές τύπου J είναι εντολές που πραγματοποιούνε άλματα στο κώδικα αλλάζοντας τη σειριακή εκτέλεσή του. Οι συγκεκριμένες εντολές προκαλούν συχνά αλλαγές στον PC (program counter) ο οποίος δείχνει σε κάθε κύκλο μηχανής την επόμενη προς εκτέλεση εντολή. Ο τύπος των εντολών αυτών δεν αποθηκεύει το αποτέλεσμα της πράξης που γίνεται πουθενά αλλού εκτός του PC, ενώ μπορεί να πάρει σαν όρισμα ένα καταχωρητή. Η παραπάνω αναφορά στο σύνολο εντολών του MIPS κρίνεται αναγκαία σε περιπτώσεις που ο χρήστης έχει τη δυνατότητα να κάνει αλλαγές στις εντολές του προγράμματος που εκτελείται στον επεξεργαστή.

Συγκεντρωτικά οι εντολές του MIPS φαίνονται στο παρακάτω πίνακα:

Εντολές επικοινωνίας με μνήμη	**Σημασία**
lb, lbu, sb	Φόρτωση byte με/χωρίς πρόσημο, αποθήκευση byte
Lh, lhu, sh	Φόρτωση λέξης με/χωρίς πρόσημο, αποθήκευση λέξης
Lw, sw	Φόρτωση/αποθήκευση λέξης (καταχωρητής ακεραίων)
Lwcl, ldcl, swcl, swcl	Φόρτωση καταχωρητή κινητής υποδιαστολής απλής/διπλής ακρίβειας, αποθήκευση καταχωρητή κινητής υποδιαστολής απλής/διπλής ακρίβειας
Αριθμητικές/λογικές εντολές	**Πράξεις σε ακέραια/λογικά δεδομένα – Καταχωρητές γενικής χρήσης**
Add, addi, addu, addui	Πρόσθεση, απευθείας πρόσθεση (16-bits) με/χωρίς πρόσημο
Sub, subi, subu, subui	Αφαίρεση, απευθείας αφαίρεση με/χωρίς πρόσημο
Mult, multu, div, divu	Πολλαπλασιασμός/διαίρεση με/χωρίς πρόσημο (καταχωρητές κινητής υποδιαστολής 32-bits)
And, andi	AND, απευθείας AND
Or, ori, xor, xori	OR, απευθείας OR, XOR, απευθείας XOR
Lui	Απευθείας φόρτωση τιμής στα δύο πιο σημαντικά bytes ενός καταχωρητή 16-bits
Sslv, srlv, srav, sll, srl, sra	Ολισθήσεις απευθείας/μεταβλητές, αριστερές/δεξιές λογικές, δεξιές αριθμητικές
Slt	Αριστερή ολίσθηση
Διακλαδώσεις	**Με συνθήκη ή μεταπηδήσεις – Σχετικές με PC ή καταχωρητές**
Beq, bne	Διακλάδωση με ίσο/άνισο περιεχόμενο μεταξύ δύο καταχωρητών
J, jr	Μεταπηδήσεις 26-bits offset από PC+4, στόχος σε καταχωρητή
Jal, jalr	Μεταπήδηση και σύνδεση, PC+4 στον $31, στόχος PC-σχετικό ή καταχωρητής
Break	Μεταφορά στο λειτουργικό σε ένα διάνυσμα διεύθυνσης
Rfe	Επιστροφή στο κώδικα από εξαίρεση

Πίνακας 3.1 Οι εντολές του MIPS συγκεντρωτικά

3.7 Η διευθυνσιοδότηση ενός MIPS

Κατά κύριο λόγο οι τρόποι διευθυνσιοδότησης ενός MIPS είναι τρείς:

α) απευθείας (immediate)

β) μέσω καταχωρητή (register addressing)

γ) έμμεσος με καταχωρητή (περιεχόμενο καταχωρητή + μία σταθερά)

Το συγκεκριμένο σύστημα διαθέτει 32 καταχωρητές γενικού σκοπού των 32-bits ($0,…, $31) με τον $0 να έχει πάντα τη τιμή 0. Περιέχει επιπλέον 32 καταχωρητές για δεδομένα κινητής υποδιαστολής ($f0,…, $f31) με 16 από αυτούς να χρησιμοποιούνται για δεδομένα μονής ακρίβειας ($f0, $f2,…, $f30). Για δεδομένα κινητής υποδιαστολής διπλής ακρίβειας οι 16 συγκεκριμένοι καταχωρητές σχηματίζουν οκτώ ζευγάρια που χρησιμοποιούνται για τους υπολογισμούς. Οι υπόλοιποι ($f1, $f3,…, $f31) χρησιμοποιούνται για φόρτωση ή αποθήκευση των λιγότερων σημαντικών ψηφίων σε αριθμούς κινητής υποδιαστολής των 64-bits. Ο MIPS ορίζει ακόμα ένα καταχωρητή τον $gp (global pointer) ο οποίος χρησιμοποιείται για να δείχνει το κομμάτι της μνήμης που περιέχει τα στατικά δεδομένα/μεταβλητές, ένα δείκτη σωρού ($sp – stack pointer), ένα δείκτη πλαισίου διαδικασίας ($fp – frame pointer) και μία διεύθυνση επιστροφής ($ra). Ο καταχωρητής $1 ($at) χρησιμοποιείται αποκλειστικά από το συμβολομεταφραστή (assembler) και οι καταχωρητές 26 – 27 ($k0 - $k1) χρησιμοποιούνται αποκλειστικά από το λειτουργικό σύστημα.

Οι τύποι δεδομένων σε ένα MIPS είναι:

α) 8-bits, 16-bits (half word), 32-bits (word) για ακέραια δεδομένα

β) 32-bits απλής ακρίβειας, 64-bits διπλής ακρίβειας για κλασματικούς αριθμούς (αριθμούς κινητής υποδιαστολής)

Τα πεδία των εντολών είναι διαφορετικά ανάλογα με το είδος της εντολής που κωδικοποιείται και με τον τρόπο διευθυνσιοδότησης που χρησιμοποιείται. Παρακάτω βλέπουμε τι ισχύει σαν γενικός κανόνας για τις βασικές εντολές:

α) Για εντολές R-τύπου

0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

β) Για εντολές load/store

35ή43	rs	rt	address
31:26	25:21	20:16	15:0

γ) Για εντολή διακλάδωσης

4	rs	rt	address
31:26	25:21	20:!6	15:0

Ο καταχωρητής προορισμού θα βρίσκεται ή στη θέση 20:16 (rt) για load, ή στη θέση 15:11 (rd) για εντολή R-τύπου. Ο καθορισμός του συγκεκριμένου καταχωρητή γίνεται και με τη βοήθεια ενός πολυπλέκτη.

Μια γενικότερη εικόνα του χάρτη μνήμης θα ήταν

0x0040-0000 .text 0x0FFF-FFFF
0x1001-0000 .data (static)
(dynamic data) .stack 0x7FFF-FFFF
0x8000-0000 .ktext (kernel) 0x8FFF-FFFF
0x9000-0000 .kdata (kernel) 0xFFFE-FFFF

Πίνακας 3.2 Χάρτης μνήμης

Ας δούμε μερικά παραδείγματα σχετικά με το πώς διευθυνσιοδοτούνται και αποκωδικοποιούνται βασικές εντολές ενός MIPS, ώστε να κατανοήσουμε καλύτερα τον τρόπο λειτουργίας του.

α) Αν έχουμε τις εντολές *and rd,rs,rt* και *add rt,rs,imm* τότε η λύση θα είναι

Στη πρώτη περίπτωση θα έχουμε κωδικοποίηση με καταχωρητή και να εκτελείται η πράξη rd<= rs and rt

op	rs	rt	rd	0	funct(and)

Στη δεύτερη περίπτωση θα έχουμε απευθείας διευθυνσιοδότηση ως εξής

op	rs	rt	immediate

β) Έστω ότι έχουμε τις εντολές *lw $1,100($0)* και sw *$1,100($2)* τότε

Έχουμε ως τρόπο κωδικοποίησης τον έμμεσο καταχωρητή με μετατόπιση. Η μετατόπιση είναι η σταθερά 100 και βρίσκεται στο τελευταίο πεδίο της εντολής. Η διεύθυνση από την οποία θα φορτωθούν τα δεδομένα στο καταχωρητή $1 υπολογίζεται από τη πρόσθεση του περιεχομένου του $0 με το 100.

op	$0	$1	100

Στη δεύτερη περίπτωση έχουμε πάλι έμμεσο καταχωρητή με μετατόπιση. Και εδώ η μετατόπιση είναι 100 και βρίσκεται στο τελευταίο πεδίο της εντολής. Με την εντολή αυτή το περιεχόμενο του $1 αποθηκεύεται σε μία θέση μνήμης, η διεύθυνση της οποίας υπολογίζεται με το προηγούμενο τρόπο.

Τα δεδομένα ονομάζονται ευθυγραμμισμένα εάν η διεύθυνση της μνήμης που τα περιέχει είναι πολλαπλάσια του μεγέθους των δεδομένων σε bytes. Οι περισσότερες εντολές load/store εκτελούνται σε τέτοια δεδομένα.

γ) Έστω οι εντολές *beq rs,rt,label* και *j label*

Στη πρώτη περίπτωση γίνεται σύγκριση των rs, rt και αν είναι ίδιο το περιεχόμενό τους τότε η διεύθυνση της επόμενης εντολής υπολογίζεται με αύξηση του PC κατά 4 + τη σταθερά label. Αν το περιεχόμενο των καταχωρητών δεν είναι ίσο τότε η επόμενη προς εκτέλεση εντολή βρίσκεται στη διεύθυνση μνήμης PC+4.

op	rs	rt	label

Με τη δεύτερη εντολή αλλάζει η ροή του προγράμματος και πηγαίνει στη διεύθυνση label. Τα πρώτα 6-bits αποτελούν το κωδικό λειτουργίας της εντολής και τα υπόλοιπα 26 το πεδίο της διεύθυνσης.

op	label

δ) Έστω οι εντολές *lw $24,AddrConstant4($0)* και *add $29,$29,$24* να τρέχουν μαζί

Εδώ γίνεται πρόσθεση της σταθεράς 4 που βρίσκεται στη διεύθυνση AddrConstant4 στο καταχωρητή $29. Αρχικά φορτώνεται η σταθερά στο καταχωρητή $24, έπειτα προστίθεται το περιεχόμενο του καταχωρητή $24 με το περιεχόμενο του καταχωρητή $29 και το αποτέλεσμα αποθηκεύεται στον $29.

Προκειμένου να αποφύγουμε εντολές επικοινωνίας με τη μνήμη (load/store) είναι προτιμότερο να χρησιμοποιήσουμε τον απευθείας τρόπο διευθυνσιοδότησης με τη βοήθεια της εντολής addi (add immediate) ως

addi $29,$29,4

ε) Έστω η εντολή *lw $1,100($2)*

Εδώ χρησιμοποιείται ο έμμεσος τρόπος διευθυνσιοδότησης με καταχωρητή. Η εκτέλεση της συγκεκριμένης εντολής μεταφέρει δεδομένα από μία θέση μνήμης σε ένα καταχωρητή. Η διεύθυνση της μνήμης που υπάρχουν τα δεδομένα υπολογίζεται από το άθροισμα της τιμής 100 και το περιεχόμενο του καταχωρητή $2, με το αποτέλεσμα να αποθηκεύεται στο καταχωρητή $1.

4. ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΟΥ ΠΡΩΤΟΤΥΠΟΥ ΕΛΕΓΧΟΥ

4.1 Περιγραφή

Για τον πρώτο έλεγχο του ολοκληρωμένου επεξεργαστή και της μνήμης έγινε μια μικρή προσθήκη δύο αριθμών.

Αρχικά η μνήμη πρέπει να φορτωθεί με τις εντολές και τα δεδομένα με χρήση αρχείων .mif για να γίνει εγγραφή των πληροφοριών στα μπλοκ της μνήμης πριν ξεκινήσει η προσομοίωση.

Οι εντολές που έχουν εγγραφεί στη μνήμη είναι:

Memory Address	**Instruction**	**Instruction Field**					
		op	**rs**	**rt**	**rd**	**shamt**	**funct**
000	lw $s0, 128($zero)	100011	00000	10000	0000000010000000		
004	lw $s1, 132($zero)	100011	00000	10001	0000000010000100		
008	add $s2, $s0, $s1	000000	10000	10001	10010	00000	100000
012	sw $s2, 136($zero)	101011	00000	10010	0000000010001000		
016	sub $s3, $s1, $s0	000000	10001	10000	10011	00000	100010
020	sw $s3, 140($zero)	101011	00000	10011	0000000010001100		
024	and $s4, $s1, $s0	000000	10001	10000	10100	00000	100100
028	sw $s4, 144($zero)	101011	00000	10100	0000000010010000		
032	or $s5, $s1, $s0	000000	10001	10000	10101	00000	100101

036	sw $s5, 148($zero)	101011	00000	10101	0000000010010100		
040	slt $s6, $s1, $s0	000000	10000	10001	10110	00000	101010
044	sw $s6, 152($zero)	101011	00000	10110	0000000010011000		
048	beq $s0, $s4, 56	000100	10000	10100	0000000000000001		
052	UNDEFINED	UUUUUU	UUUUU	UUUUU	UUUUU	UUUUU	UUUUU
056	j 8	000010	00000000000000000000000010				

Πίνακας 4.1 Εντολές που δίνονται

Τα δεδομένα που γράφτηκαν στη μνήμη είναι:

Memory Address	Data (dec)	Data (bin)
128	379	00000000 00000000 00000001 01111011
132	383	00000000 00000000 00000001 01111111

Πίνακας 4.2 Δεδομένα που γράφονται στη μνήμη

Οι αναμενόμενες τιμές που αποθηκεύτηκαν πίσω στη μνήμη είναι:

Memory Address	Data (dec)	Data (bin)
136	762	00000000 00000000 00000010 11111010
140	4	00000000 00000000 00000000 00000100
144	379	00000000 00000000 00000001 01111011
148	383	00000000 00000000 00000001 01111111
152	1	00000000 00000000 00000000 00000001

Πίνακας 4.3 Τιμές που αποθηκεύονται πίσω στη μνήμη

Η προσομοίωση ξεκινά στη διεύθυνση μνήμης 000 με φόρτωση οδηγίας λέξης. Η τιμή της διεύθυνση μνήμης 128 γράφεται στο καταχωρητή $s0. Ο PC αυξάνεται και η εντολή στη διεύθυνση μνήμης 004 εκτελείται. Υπάρχει επίσης και μια εντολή φόρτωσης λέξης που αποθηκεύει τη τιμή της διεύθυνσης 132 στο καταχωρητή $s1.

Ακολουθεί μια εντολή πρόσθεσης η οποία προσθέτει τους όρους που είναι γραμμένοι στου καταχωρητές $s0, $s1 και γράφει το αποτέλεσμα στο καταχωρητή $s2.

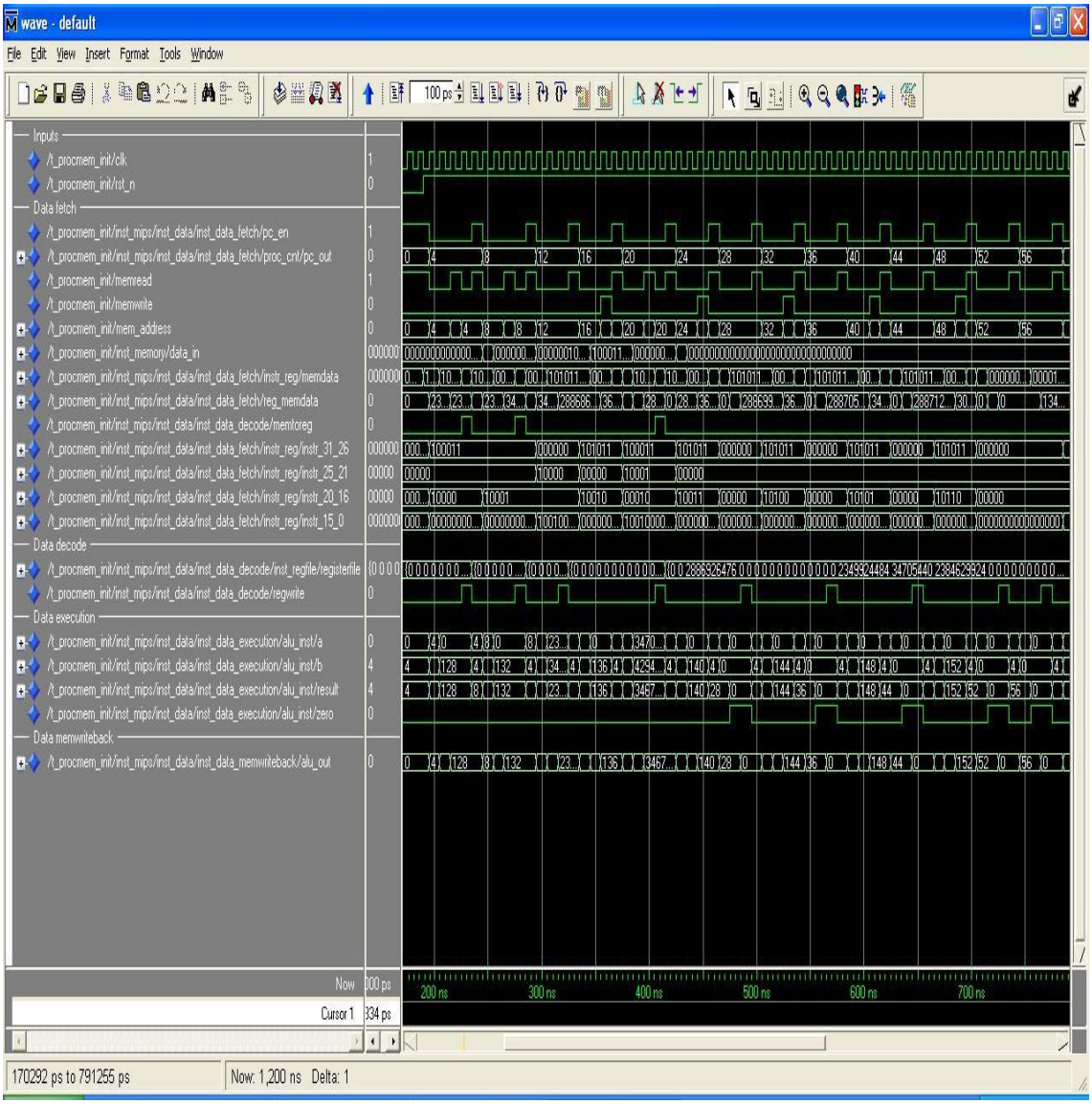
Τότε μια εντολή αποθήκευσης λέξης γράφει το περιεχόμενο του $s2 στη μνήμη, στη διεύθυνση 136.

Οι επόμενες εντολές είναι για αφαίρεση, πρόσθεση, λογική πράξη or, beq και jump. Το αποτέλεσμα ενός υπολογισμού καταχωρείται πάντα στη μνήμη με μια εντολή αποθήκευσης λέξης.

Σημείωση:

Οι εντολές του assembler που χρησιμοποιούνται δε καθορίζονται πλήρως σε αυτή την αναφορά. Για πληροφορίες σχετικά με τη γλώσσα μηχανής δείτε το σύγγραμμα των Patterson – Hennesy, ‘Computer Organization and Design’[1] κεφάλαια 2, 3.

4.2 Αποτελέσματα προσομοίωσης



Εικόνα 4.1 Αποτελέσματα προσομοίωσης MIPS και μνήμης

5. ΕΝ ΚΑΤΑΚΛΕΙΔΙ

5.1 Οι εμπειρίες μου

Δουλεύοντας πάνω στη συγκεκριμένη εργασία έγινε περισσότερο κατανοητή η λογική βάσει της οποίας μπορεί να λειτουργήσει ένα υπολογιστικό σύστημα. Ιδιαίτερα αναλύθηκε ο τρόπος λειτουργίας ενός επεξεργαστή MIPS και μια πιθανή αναπαράστασή του μέσω της γλώσσας VHDL και του πακέτου Altera Quartus. Η θεωρητική προσέγγιση του έργου συμπλήρωσε με έγκυρο τρόπο τη γνώση γύρω από τη φιλοσοφία των επεξεργαστών και απέδωσε τμηματικά την εικόνα αυτού του μικρόκοσμου που συνιστά την «καρδιά» και τον «εγκέφαλο» της τεχνολογίας του σήμερα.

6. ΑΝΑΦΟΡΕΣ

[1] David A. Patterson, John L. Hennessy: 'Computer Organization and Design - The Hardware/Software Interface" Second Editon (1998) Morgan Kaufmann Publisher, Inc.

[2] Altera Corporation Cyclone II FPGA Family Erata Sheet ES-030405-1.3 www.altera.com

[3] www.wikipedia.com/vhdl

[4] https://forums.xilinx.com/t5/General-Technical-Discussion/registereoutput-and-registered-input/td-p/341981

[5] David W. Wall 'Limits of instruction level parallelism' Volume19

[6] http://www.cs.utexas.edu/users/cart/publications/isca00.pdf

[7] https://lca.ece.utexas.edu/pubs/spec09_ciji.pdf

[8] http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

[9] http://www.hs-augsburg.de/~mlinder/Files/Proc_Implementation_VHDL.pdf

[10] http://www.cim.mcgill.ca/~langer/273/16-notes.pdf

[11]http://home.deib.polimi.it/silvano/FilePDF/ARCMULTIMEDIA/mipsr4000_00127580.pdf

[12] http://american.cs.ucdavis.edu/publications/p234-farrens.pdf

[13] http://esd.cs.ucr.edu/labs/tutorial/

[14]http://www.eng.ucy.ac.cy/mmichael/courses/ECE314/LabsNotes/02/MIPS_Instruction_Coding_With_Hex.pdf

[15]http://eclass.uth.gr/eclass/modules/document/file.php/MHX111/%CE%94%CE%B9%CE%B4%CE%B1%CE%BA%CF%84%CE%B9%CE%BA%CF%8C%20%CE%A0%CE%B1%CE%BA%CE%AD%CF%84%CE%BF/Lec2_Assembly_I.pdf

[16]https://classes.soe.ucsc.edu/cmpe110/Spring11/lectures/05_MIPS_addressing.pdf

[17]https://opencourses.uoc.gr/courses/pluginfile.php/11668/mod_resource/content/1/Ask3.pdf

[18] http://arch.icte.uowm.gr/courses/arch/oc_archlab-theory-03.pdf

[19] http://newtech-pub.com/wpcontent/uploads/2013/10/Arithmirtika_Kefalaio-1.pdf

[20] https://forums.xilinx.com/t5/GeneralTechnicalDiscussion/registeredoutput-and-registered-input/td-p/341981

[21]http://electronics.stackexchange.com/questions/210999/quartuswhatisthe-purpose-to-register-output-port-when-ram-or-rom-megawizar