



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ**

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

**ΣΧΕΔΙΑΣΗ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ  
ΕΠΕΞΕΡΓΑΣΤΗ RISC ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ  
LOGISIM**

**Διπλωματική Εργασία**

Αναστάσιος Βουρναζίδης

**Επιβλέπων καθηγητής**

Αριστείδης Ευθυμίου

**ΙΩΑΝΝΙΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2020**



# Κατάλογος περιεχομένων

<b>Κεφάλαιο 1ο:Εισαγωγή.....</b>	<b>7</b>
1.1 Αρχιτεκτονική RISC.....	7
1.2 Ιστορία της RISC.....	8
1.3 Χαρακτηριστικά της RISC.....	9
1.4 RISC vs. CISC.....	10
<b>Κεφάλαιο 2ο:Αρχιτεκτονική MIPS.....</b>	<b>12</b>
2.1 Εισαγωγή.....	12
2.2 Σύνολο εντολών.....	13
2.3 Μία βασική υλοποίηση επεξεργαστή MIPS.....	14
2.3.1 Μορφή εντολών.....	14
2.3.2 Διαδρομή δεδομένων.....	16
2.3.3 Έλεγχος.....	20
2.3.4 Διοχέτευση.....	25
<b>Κεφάλαιο 3ο:Σχεδίαση και προσομοίωση του MIPS με τη χρήση του Logisim.....</b>	<b>31</b>
3.1 Εισαγωγή.....	31
3.2 Σχεδιασμός επεξεργαστή MIPS στο Logisim .....	31
3.2.1 Στάδιο προσκόμισης εντολής.....	31
3.2.2 Στάδιο αποκωδικοποίησης εντολής & ανάγνωσης καταχωρητών.....	33
3.2.3 Στάδιο εκτέλεσης λειτουργίας ή υπολογισμός διεύθυνσης.....	36
3.2.4 Προσπέλαση τελεστέου μνήμης.....	38
3.2.5 Εγγραφή αποτελέσματος.....	39
3.2.6 Ολοκλήρωση του επεξεργαστή.....	39
3.3 Προσομοίωση.....	40
<b>Κεφάλαιο 4ο:Συμπεράσματα.....</b>	<b>46</b>
<b>Βιβλιογραφία.....</b>	<b>49</b>

# **Ευχαριστίες**

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της εργασίας μου κ. κ. Αριστείδη Ευθυμίου, επίκουρο καθηγητή του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων για την άψογη συνεργασία, την αμέριστη βοήθεια και για τις γνώσεις που μου μετέδωσε κατά την διάρκεια υλοποίησης της παρούσας εργασίας.

Φεβρουάριος 2020

Αναστάσιος Βουρναζίδης

# Περίληψη

Στην παρούσα εργασία περιγράφεται η διαδικασία σχεδίασης ενός επεξεργαστή MIPS καθώς και η φιλοσοφία πίσω από αυτή, χρησιμοποιώντας το Logisim, ένα εργαλείο για σχεδίαση και προσομοίωση ψηφιακών κυκλωμάτων. Αρχικά, γίνεται μια περιγραφή της ανάπτυξης, εξέλιξης και των χαρακτηριστικών των επεξεργαστών RISC ώστε να γίνει κατανοητός ο τρόπος λειτουργίας τους. Στη συνέχεια, σχεδιάζουμε και αναλύουμε στη θεωρία μια απλή μορφή ενός επεξεργαστή MIPS ενός κύκλου ρολογιού με διοχέτευση. Έπειτα, υλοποιούμε αυτή τη σχεδίαση στο πρόγραμμα Logisim. Με το εν λόγω εργαλείο προσομοιώνουμε την λειτουργία και τις επιδόσεις του επεξεργαστή, εμβαθύνοντας την κατανόηση μας σχετικά με τον τρόπο λειτουργίας του, δίνοντάς μας την ευκαιρία να πειραματιστούμε, ενώ μας προσφέρει τη δυνατότητα για περαιτέρω βελτίωση και εξέλιξη. Τέλος, γίνεται μια εκτίμηση των πλεονεκτημάτων και των μειονεκτημάτων της συγκεκριμένης υλοποίησης σε σχέση με υλοποιήσεις με χρήση άλλων εργαλείων.

**Λέξεις Κλειδιά:** επεξεργαστής, RISC, MIPS, Logisim, σχεδίαση, ανάπτυξη, προσομοίωση

# Abstract

This paper describes the design process of a MIPS processor as well as the philosophy behind it, using Logisim, an educational tool for designing and simulating digital logic circuits. Initially, a description of the development, evolution and characteristics of RISC processors is provided to understand how they work. Next, we design and analyze in theory a simple form of a pipelined single clock cycle MIPS processor. Afterwards, we implement this design in the Logisim program. This tool simulates the proper operation and performance of the processor, deepening our understanding of how the processor operates, giving us the opportunity to experiment, thus giving us the potential for further improvement and development. Finally, an appraisal is made of the advantages and disadvantages of this embodiment in relation to embodiments using other tools.

**Keywords:** processor, RISC, MIPS, Logisim, design, development, simulation

# Κεφάλαιο 1ο: Εισαγωγή

Η σχεδίαση ενός επεξεργαστή είναι μία περίπλοκη διαδικασία ωστόσο ακολουθώντας κάποια προκαθορισμένα βήματα η υλοποίηση μπορεί να απλοποιηθεί σε κάποιο βαθμό. Η διαδικασία σχεδιασμού περιλαμβάνει την επιλογή ενός συνόλου εντολών και ενός ορισμένου παραδείγματος εκτέλεσης (π.χ. RISC ή CISC) και οδηγεί σε μια μικροαρχιτεκτονική, η οποία μπορεί να περιγραφεί από γλώσσες προγραμματισμού όπως η VHDL ή Verilog. Ο τρόπος λειτουργίας του κάθε επεξεργαστή είναι η εκτέλεση λιστών εντολών. Μέσα στις εντολές περιλαμβάνονται, συνήθως, όλες οι πληροφορίες που χρειάζονται για τον υπολογισμό ή την επεξεργασία των δεδομένων με τη χρήση καταχωρητών, την αλλαγή, αποθήκευση ή φόρτωση από την μνήμη τιμών δεδομένων καθώς και εκτέλεση σχεσιακών πράξεων για τον έλεγχο της ροής του προγράμματος.

Για να γίνει πλήρως κατανοητή η σχεδίαση θα κάνουμε μία εισαγωγή στους επεξεργαστές ξεκινώντας από την αρχιτεκτονική RISC στην οποία ανήκουν και οι επεξεργαστές αρχιτεκτονικής MIPS. Έπειτα θα αναλύσουμε την ίδια την αρχιτεκτονική MIPS και έπειτα θα ξεκινήσουμε την σχεδίαση του επεξεργαστή MIPS θεωρητικά και πρακτικά. Η πρακτική πλευρά θα υλοποιηθεί με τη χρήση του προγράμματος προσομοίωσης Logisim. Η επιλογή του συγκεκριμένου προγράμματος έγινε επειδή οι προσομοιώσεις του γίνονται πιο άμεσα καθώς σε κάθε βήμα εκτέλεσης μπορούμε να δούμε τι τιμές έχει κάθε καταχωρητής, κάθε καλώδιο και γενικά κάθε μεταβλητή του κυκλώματος. Επίσης, υποστηρίζει και τη χρήση εξαρτημάτων VDHL, που απλοποιεί αρκετά κάποια σημεία του επεξεργαστή που θα ήταν περίπλοκα και χρονοβόρα να υλοποιηθούν με λογικά κυκλώματα.

Τέλος, αφού σχεδιάσουμε και προσομοιώσουμε τον επεξεργαστή θα έχουμε μία ολοκληρωμένη άποψη σχετικά με το βαθμό δυσκολίας της χρήσης του Logisim σε σχέση με το ModelSim και θα μπορούμε να βγάλουμε χρήσιμα συμπεράσματα σχετικά με τα πλεονεκτήματα και μειονεκτήματα της υλοποίησης και προσομοίωσης του κάθε προγράμματος, καθώς και τις επιλογές που προσφέρει το καθένα.

## 1.1 Αρχιτεκτονική RISC

RISC, ή Redused Instruction Set Computer (Υπολογιστής Μειωμένου Συνόλου

Εντολών), είναι ένας τύπος αρχιτεκτονικής μικροεπεξεργαστών που χρησιμοποιεί ένα μικρό, άκρως βελτιστοποιημένο σύνολο εντολών και όχι ένα πιο εξειδικευμένο σύνολο εντολών που συχνά συναντώνται σε άλλους τύπους αρχιτεκτονικών. Η αρχιτεκτονική του συνόλου εντολών του (ISA – instruction set architecture), του επιτρέπει να έχει λιγότερους κύκλους ανά εντολή (CPI) από έναν υπολογιστή σύνθετων εντολών (CISC). Έχουν γίνει διάφορες προτάσεις σχετικά με τον ακριβή ορισμό του RISC, αλλά η γενική ιδέα είναι ότι ένας τέτοιος υπολογιστής έχει ένα μικρό σύνολο απλών και γενικών εντολών και όχι ένα μεγάλο σύνολο σύνθετων και εξειδικευμένων εντολών. Ωστόσο, οι επεξεργαστές RISC μοιράζονται ένα πλήθος κοινών χαρακτηριστικών τα οποία θα παρουσιάσουμε και θα αναλύσουμε στην συνέχεια.

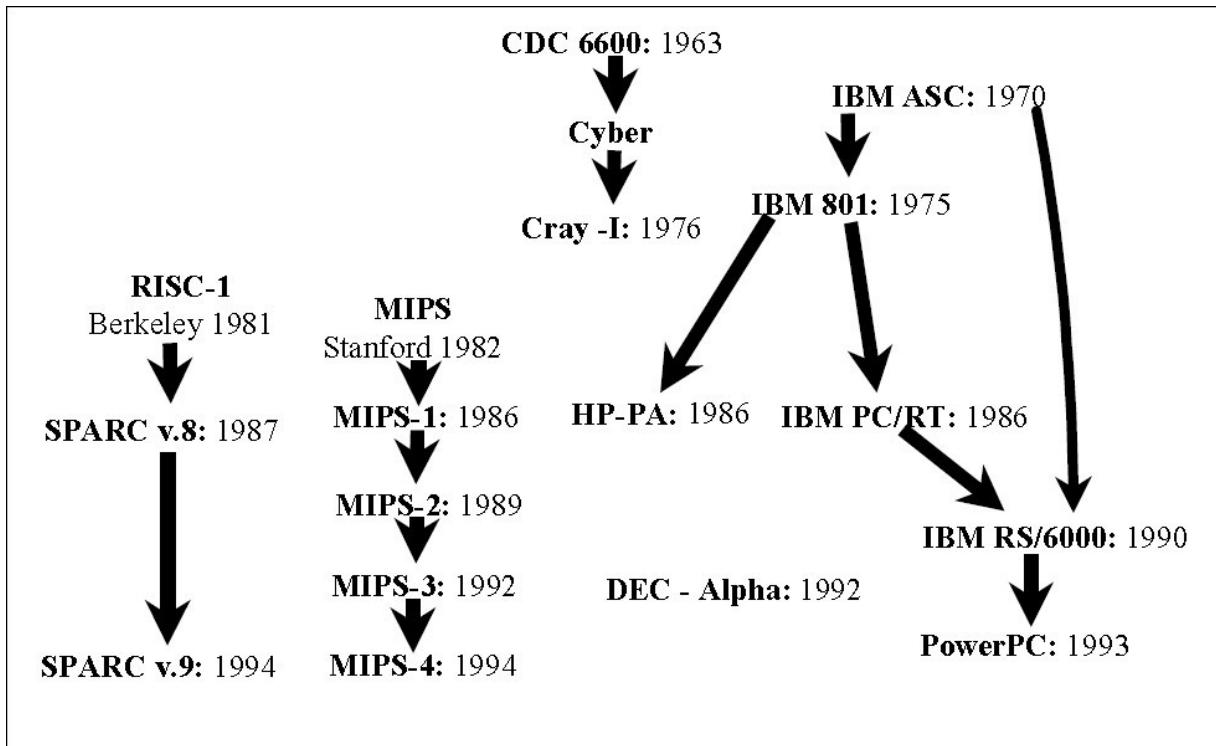
## 1.2 Ιστορία της RISC

Η αρχιτεκτονική RISC προέκυψε ως μια μακρά και εξελικτική διαδικασία στην ιστορία της ανάπτυξης ηλεκτρονικών υπολογιστών στην οποία μάθαμε πώς να χτίσουμε καλύτερα και πιο αποτελεσματικά υπολογιστικά συστήματα. Το πρόγραμμα RISC ξεκίνησε το 1975 στο ερευνητικό κέντρο T. J. Watson, της IBM, υπό την ονομασία 801 και ήταν άγνωστο εκτός της IBM μέχρι τις αρχές της δεκαετίας του '80. Ο IBM 801 κατασκευάστηκε τελικά σε μορφή μονού μικροκυκλώματος (single-chip) το 1981.

Η ιδέα ενός απλούστερου υπολογιστή, ειδικά αυτού που μπορεί να εφαρμοστεί σε μονό μικροκύκλωμα, ήταν ελκυστική και ξεκίνησαν δύο άλλα προγράμματα με παρόμοιους στόχους στις αρχές της δεκαετίας του 1980 στο Πανεπιστήμιο της Καλιφόρνιας Berkeley (RISC) και στο Πανεπιστήμιο του Στάνφορντ (MIPS). Το Berkeley RISC βασιζόταν στην αύξηση της απόδοσης μέσω της χρήσης διοχέτευσης και μιας τεχνικής που ονομάζεται παραθύρωση καταχωρητών. Το πρόγραμμα MIPS που εξελίχθηκε από ένα μεταπτυχιακό μάθημα από τον John L. Hennessy στο Πανεπιστήμιο του Στάνφορντ το 1981, κατέληξε σε ένα λειτουργικό σύστημα το 1983 και μπορούσε να εκτελέσει απλά προγράμματα μέχρι το 1984. Η προσέγγιση MIPS εστίαζε σε επιθετικό κύκλο ρολογιού και τη χρήση της διοχέτευσης, διασφαλίζοντας ότι θα μπορούσε να λειτουργεί όσο το δυνατόν πληρέστερα.

Στο παρακάτω χρονογράφημα απεικονίζεται η ανάπτυξη της αρχιτεκτονικής

RISC:



Εικόνα 1.1: Η Ιστορία της ανάπτυξης της αρχιτεκτονικής RISC

## 1.3 Χαρακτηριστικά της RISC

Όπως αναφέρθηκε και στην αρχή του κεφαλαίου ο όρος RISC είναι ασαφής και η κεντρική ιδέα είναι ότι ένα σύστημα RISC έχει ένα μικρό σύνολο απλών και γενικών εντολών. Υπάρχουν, όμως κι άλλα στοιχεία που χαρακτηρίζουν τα συστήματα RISC:

1. Διοχέτευση (Pipelining): Αποτελεί τυπικό χαρακτηριστικό σε επεξεργαστές RISC και μοιάζει αρκετά με γραμμή συναρμολόγησης. Επειδή ο επεξεργαστής λειτουργεί ταυτόχρονα σε διαφορετικά βήματα της εντολής, περισσότερες οδηγίες μπορούν να εκτελεστούν παράλληλα, άρα και σε μικρότερο χρονικό διάστημα.
2. Μια εντολή ανά κύκλο: Ένας κύκλος μηχανής είναι ο χρόνος που απαιτείται για να ληφθούν δύο τελεστέοι από τους καταχωρητές, να εκτελεστεί η πράξη στην ALU με αυτούς και να αποθηκευτεί το αποτέλεσμα σε έναν καταχωρητή. Επομένως, η εκτέλεση εντολών RISC διαρκεί περίπου όσο και οι μικροεντολές (micro-instructions) στα συστήματα CISC.
3. Καταχωρητής σε καταχωρητή τελεστέοι: Στα συστήματα RISC οι πράξεις που

προσπελαύνει τη μνήμη είναι η φόρτωση και η αποθήκευση. Όλοι οι άλλοι τελεστέοι διατηρούνται σε καταχωρητές. Αυτό το σχεδιαστικό χαρακτηριστικό απλοποιεί το σύνολο εντολών και επομένως απλοποιεί τη μονάδα ελέγχου.

4. Απλές λειτουργίες διευθυνσιοδότησης: Ένα άλλο χαρακτηριστικό είναι η χρήση απλών τρόπων διευθυνσιοδότησης. Οι μηχανές RISC χρησιμοποιούν απλή διευθυνσιοδότηση καταχωρητή, ενώ πιο περίπλοκοι τρόποι συνθέτονται από το λογισμικό από αυτές τις απλές λειτουργίες.
5. Απλές μορφές εντολών: Το RISC χρησιμοποιεί απλές μορφές εντολών. Γενικά, χρησιμοποιούνται μόνο μία ή λίγες μορφές εντολών. Σε τέτοια συστήματα το μήκος της εντολής είναι σταθερό και ευθυγραμμίζεται με όρια λέξεων.
6. Απόδοση με χρήση βέλτιστων μεταγλωττιστών: Καθώς οι οδηγίες είναι απλές, οι μεταγλωττιστές μπορούν να αναπτυχθούν για αποτελεσματική οργάνωση κώδικα, μεγιστοποιώντας επίσης τη χρήση καταχωρητών κλπ.
7. Υψηλή απόδοση εκτέλεσης εντολών: Κατά τη χαρτογράφηση γλώσσας υψηλού επιπέδου σε εντολές μηχανής ο μεταγλωττιστής ευνοεί τις σχετικά απλές οδηγίες. Οι απλές οδηγίες υποστηρίζουν καλύτερες δυνατότητες χρήσης της διοχέτευσης εντολών.

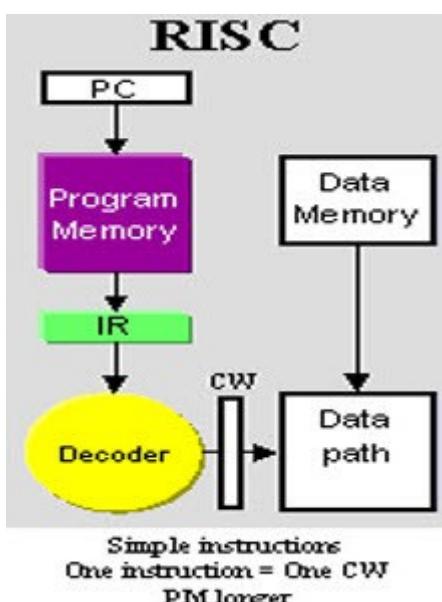
## 1.4 RISC vs. CISC

Η αρχιτεκτονική CISC, ή αλλιώς, Complex Instruction Set Computer (Υπολογιστής Σύνθετου Συνόλου Εντολών), όπως είναι το πλήρες όνομά του αποτελεί κι αυτή μία αρχιτεκτονική συνόλου εντολών (ISA) αντίστοιχη της RISC. Η CISC έχει την ικανότητα να εκτελεί λειτουργίες πολλαπλών βημάτων ή τρόπους διευθυνσιοδότησης σε ένα σύνολο εντολών (instruction set). Είναι ο σχεδιασμός της CPU, έτσι ώστε μια εντολή εκτελεί πολλές εντολές χαμηλού επιπέδου. Ας δούμε, λοιπόν, μερικές από τις κύριες διαφορές τους:

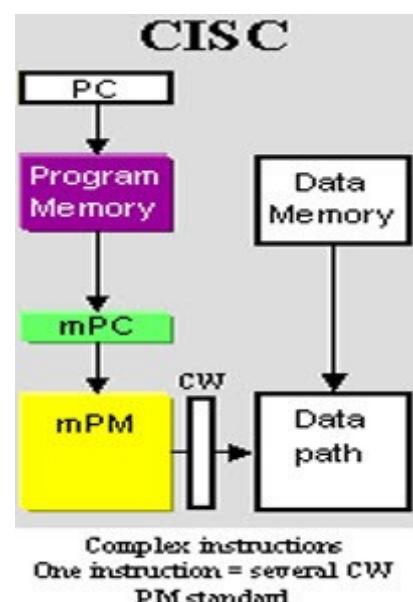
- Η απόδοση της αρχιτεκτονική RISC βελτιστοποιείται δύνοντας έμφαση στο λειτουργικό, ενώ η CISC επικεντρώνεται στο υλισμικό.
- Οι εντολές στην RISC ανήκουν στο ίδιο σύνολο με λίγες μορφές εντολών, ωστόσο υπάρχει ποικιλία εντολών τέτοια ώστε να μπορούν να υλοποιηθούν σύνθετοι υπολογισμοί. Αντίθετα, η CISC υποστηρίζει εντολές με πολλαπλά μεγέθη και

μορφές οι οποίες χρησιμοποιούνται και για τις σύνθετες εντολές.

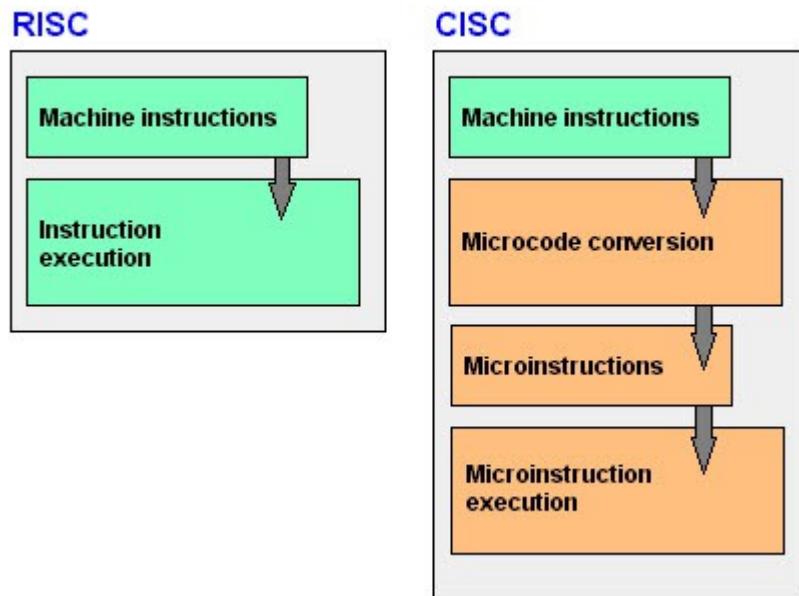
- Στην RISC οι σύνθετες λειτουργίες διευθυνσιοδότησης συνθέτονται από τις πιο απλές από το λειτουργικό. Η CISC, από την άλλη, υποστηρίζει εξ αρχής περίπλοκες λειτουργίες διευθυνσιοδότησης.
- Η CISC βασίζεται σε λιγότερους καταχωρητές, ενώ η RISC χρησιμοποιεί περισσότερους.
- Κύριο χαρακτηριστικό της RISC είναι η διοχέτευση όπως αναφέρθηκε πιο πάνω σε αντίθεση με τη CISC που συνήθως δεν χρησιμοποιεί διοχέτευση ή όχι σε τέτοιο βαθμό όσο η RISC.
- Σημαντικό μειονέκτημα για την αρχιτεκτονική RISC είναι ότι η επέκταση κώδικα μπορεί να προκαλέσει προβλήματα στην λειτουργία του επεξεργαστή και γενικότερα είναι πιο δύσκολη, γεγονός που δεν ισχύει για την CISC.



Εικόνα 1.2: Αρχιτεκτονική RISC



Εικόνα 1.3: Αρχιτεκτονική CISC



Εικόνα 1.4: Διαφορά ανάμεσα σε RISC και CISC

## Κεφάλαιο 2ο: Αρχιτεκτονική MIPS

### 2.1 Εισαγωγή

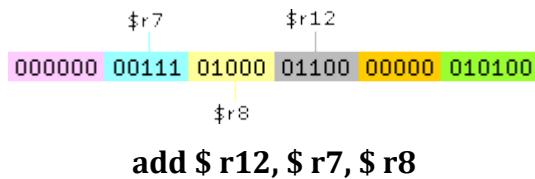
Η παρούσα εργασία αφορά τον σχεδιασμό ενός RISC επεξεργαστή και συγκεκριμένα του MIPS. Η ερευνητική ομάδα του Stanford είχε ένα ισχυρό υπόβαθρο σε μεταγλωττιστές, γεγονός που τους οδήγησε στην ανάπτυξη ενός επεξεργαστή του οποίου η αρχιτεκτονική θα αντιπροσωπεύει τη μείωση του μεταγλωττιστή σε επίπεδο υλικού, σε αντίθεση με την αύξηση του υλικού στο επίπεδο του λογισμικού, φιλοσοφία σχεδιασμού που επικρατούσε μέχρι τότε στη βιομηχανία υλικού.

Έτσι, ο επεξεργαστής MIPS υλοποιήθηκε με ένα μικρότερο, απλούστερο σύνολο εντολών. Κάθε μία από τις εντολές που περιλαμβάνονται στο σχεδιασμό του μικροκυκλώματος έτρεξε σε ένα μονό κύκλο ρολογιού. Ο επεξεργαστής χρησιμοποίησε την τεχνική της διοχέτευσης για αποτελεσματικότερη επεξεργασία των οδηγιών.

Ο MIPS χρησιμοποιεί 32 καταχωρητές, ο καθένας μήκους 32 bits (ένα μοτίβο bit αυτού του μεγέθους αναφέρεται ως λέξη - word).

## 2.2 Σύνολο εντολών

Το σύνολο εντολών MIPS αποτελείται από περίπου 111 συνολικές εντολές, καθεμία αποτελούμενη από 32 bits. Ένα παράδειγμα μιας εντολής MIPS είναι παρακάτω:



Παραπάνω απεικονίζεται η εντολή πρόσθεσης στον MIPS σε συμβολογλώσσα (πάνω) και σε δυαδική μορφή (κάτω). Η εντολή πληροφορεί τον επεξεργαστή να υπολογίσει το άθροισμα των τιμών στους καταχωρητές 7 και 8 και να αποθηκεύσει το αποτέλεσμα στον καταχωρητή 12. Τα σύμβολα του δολαρίου χρησιμοποιούνται για να υποδείξουν μια ενέργεια σε έναν καταχωρητή. Η έγχρωμη δυαδική αναπαράσταση στα δεξιά απεικονίζει τα 6 πεδία μιας εντολής MIPS. Ο επεξεργαστής προσδιορίζει τον τύπο της εντολής από τα δυαδικά ψηφία στο πρώτο και τελευταίο πεδίο (ροζ και πράσινο). Σε αυτή την περίπτωση, ο επεξεργαστής αναγνωρίζει ότι αυτή η εντολή είναι μια πρόσθεση από το μηδέν στο πρώτο του πεδίο και τον αριθμό 20 στο τελευταίο του πεδίο.

Οι τελεστέοι εμφανίζονται στο μπλε και κίτρινο πεδίο και η επιθυμητή θέση του αποτελέσματος εμφανίζεται στο τέταρτο (μοβ) πεδίο. Το πορτοκαλί πεδίο αντιπροσωπεύει την ποσότητα ολίσθησης (shift), κάτι που δεν χρησιμοποιείται σε μια πράξη πρόσθεσης.

Το σετ εντολών αποτελείται από μια ποικιλία βασικών οδηγιών, όπως:

- 21 αριθμητικές εντολές (+, -, \*, /, %),
- 8 λογικές εντολές (&, |, ~),
- 8-bit εντολές χειρισμού,
- 12 εντολές σύγκρισης (>, <, =, >=, <=, ≠),
- 25 εντολές κλάδου / άλματος (branch/jump),
- 15 εντολές φόρτωσης (load),

- 10 εντολές αποθήκευσης (store),
- 8 εντολές μετακίνησης,
- 4 ειδικές εντολές.

## 2.3 Μία βασική υλοποίηση επεξεργαστή MIPS

### 2.3.1 Μορφή εντολών

Για να υπάρξει καλύτερη κατανόηση της αρχιτεκτονικής του MIPS και του τρόπου λειτουργίας του θα περιγράψουμε μία απλή και βασική υλοποίηση επεξεργαστή MIPS.

Ας ξεκινήσουμε από το σύνολο εντολών. Το σύνολο εντολών θα έχει κάποιες κατηγορίες εντολών. Οι κατηγορίες αυτές χωρίζονται ανάλογα με την μορφή της κάθε εντολής και αποκαδικοποιούνται από την ALU, το οποίο θα αποτελεί υποσύνολο του πυρήνα του συνόλου εντολών του MIPS και περιλαμβάνει τις παρακάτω εντολές:

- Αναφοράς στη μνήμη, lw (load word) και sw (store word).
- Αριθμητικές, add, sub, and, or και slt.
- Τις beq (branch equal) και jump (j).

Οι εντολές στον MIPS, όπως και στους περισσότερους RISC επεξεργαστές, αναπαριστώνται ως λέξεις των 32 bit. Οι λέξεις χωρίζονται με τη σειρά τους σε πεδία, όπου το κάθε πεδίο περιέχει μια πληροφορία σχετική με την εντολή. Υπάρχουν 3 πρότυπα (instruction format types) με βάση τα οποία γίνεται η κωδικοποίηση των εντολών:

- I-format: για εντολές με σταθερές πχ. addi, subi κτλ., μεταφορά δεδομένων και διακλαδώσεις,
- J-format: για την εντολή j (jump),

- R-format: για όλες τις υπόλοιπες.

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

α. Εντολή τύπου R

Πεδίο	35 ή 43	rs	rt	διεύθυνση
Θέσεις bit	31:26	25:21	20:16	15:0

β. Εντολή load ή store

Πεδίο	4	rs	rt	διεύθυνση
Θέσεις bit	31:26	25:21	20:16	15:0

γ. Εντολή διακλάδωσης

**Εικόνα 2.1:** Οι τρεις κατηγορίες εντολών που θα χρησιμοποιηθούν στην υλοποίηση του MIPS, χωρισμένες σε πεδία.

Οι εντολές τύπου R έχουν κωδικό λειτουργίας (opcode) πάντα ίσο με το 0 [31:26]. Τα πεδία rs [25:21] και rt [20:16] είναι τελεστέοι καταχωρητές προέλευσης, και ο rd [15:11] προορισμού. Το πεδίο shamt [10:6] χρησιμοποιείται μόνο για ολισθήσεις. Η λειτουργία της ALU βρίσκεται στο πεδίο funct [5:0] και αποκωδικοποιείται από τη σχεδίαση της μονάδας ελέγχου ALU.

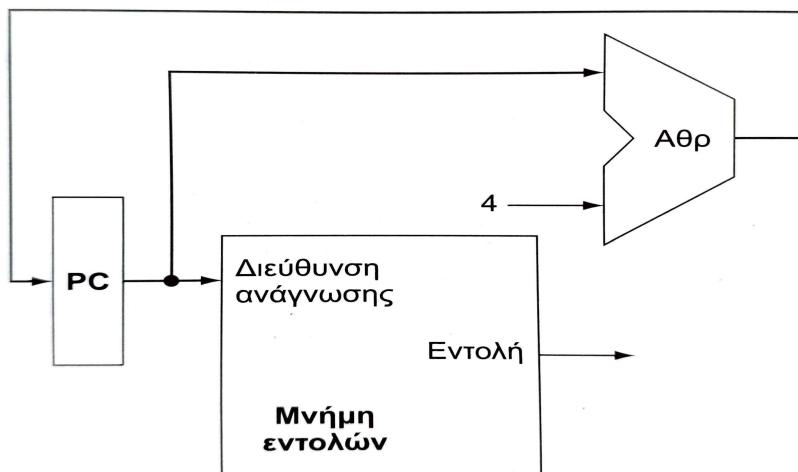
Η μορφή εντολής για εντολές με σταθερές ή μεταφοράς δεδομένων από/προς την μνήμη έχουν παρόμοια μορφή διατηρώντας τα πεδία opcode[31:26], rs[25:21] και rt[20:16]. Τα τελευταία 16 bit που απομένουν αποτελούν ένα πεδίο το οποίο μπορεί να αναπαραστήσει  $2^{16}$  τιμές, εύρος αρκετά μεγάλο για offset σε load/store word ή για σύγκριση. Για τις εντολές μεταφοράς από/προς την μνήμη ο καταχωρητής rs είναι ο καταχωρητής βάσης που προστίθεται στο πεδίο διεύθυνσης 16 bit για να σχηματιστεί η διεύθυνση μνήμης. Για τις φορτώσεις (loads), ο rt είναι ο καταχωρητής προορισμού για την τιμή που φορτώνεται. Για τις αποθηκεύσεις (stores), ο rt είναι ο καταχωρητής προέλευσης του οποίου η τιμή πρέπει να αποθηκευτεί στη μνήμη. Για τις εντολές διακλάδωσης πχ. Beq (be equal) ή bne (be not equal), οι καταχωρητές rs και rt είναι οι καταχωρητές προέλευσης που συγκρίνονται για ισότητα. Το πεδίο διεύθυνσης 16 bit υφίσταται επέκταση πρόσημου και ολίσθηση και προστίθεται στον PC για να υπολογιστεί η διεύθυνση προορισμού της διακλάδωσης.

Η εντολή άλματος j (J-format type) χρειάζεται τη δυνατότητα να πάει οπουδήποτε στο πρόγραμμα. Το ιδανικό θα ήταν μια σταθερά μήκους 32 bit. Ωστόσο, η εντολή χωρίζεται σε δύο πεδία, το opcode που παραμένουν τα 6 πιο σημαντικά bit και τα

υπόλοιπα 26 χρησιμοποιούνται για την σταθερά.

### 2.3.2 Διαδρομή δεδομένων

Αφού καθορίσαμε το σύνολο εντολών τώρα μπορούμε να ξεκινήσουμε την σχεδίαση της διαδρομής δεδομένων (datapath). Το πρώτο στοιχείο που χρειαζόμαστε είναι μία μνήμη μέσα στην οποία θα αποθηκεύονται οι εντολές του προγράμματος που εκτελείται. Η μνήμη αυτή παίρνει σαν είσοδο μία διεύθυνση και παρέχει την εντολή που βρίσκεται σε εκείνη τη θέση. Χρειαζόμαστε έναν μετρητή προγράμματος (PC) που θα κρατάει τη διεύθυνση της τρέχουσας εντολής, η υλοποίηση του οποίου γίνεται με έναν καταχωρητή. Επίσης, χρειάζεται και ένας αθροιστής για να αυξάνουμε (+4 byte) τον PC για να δείχνει στην διεύθυνση της επόμενης εντολής. Τώρα, μπορούμε να προσκομίσουμε την εντολή από την μνήμη και να αυξήσουμε τον μετρητή προγράμματος, PC, κατά 4 ώστε να δείχνει 4 byte μετά. Στην παρακάτω εικόνα φαίνονται τα τρία στοιχεία που αναφέραμε πιο πάνω και ο τρόπος που αυτά συνδέονται μεταξύ τους.



Εικόνα 2.2: Ο μετρητής προγράμματος PC, η μνήμη εντολών και ο αθροιστής για την αύξηση του PC

Όπως προείπαμε ο MIPS και γενικά οι RISC επεξεργαστές κάνουν ευρεία χρήση καταχωρητών. Γι' αυτό τον λόγο οι 32 καταχωρητές γενικού σκοπού του επεξεργαστή αποθηκεύονται στο αρχείο καταχωρητών, μία συλλογή καταχωρητών, που περιέχει την κατάσταση τους και μπορεί να γίνει ανάγνωση και εγγραφή αν πάρει σαν είσοδο έναν αριθμό καταχωρητή για προσπέλαση. Σύμφωνα με το σύνολο εντολών που ορίσαμε και τις μορφές των εντολών, οι εντολές R διαβάζουν δύο καταχωρητές για να εκτελέσουν

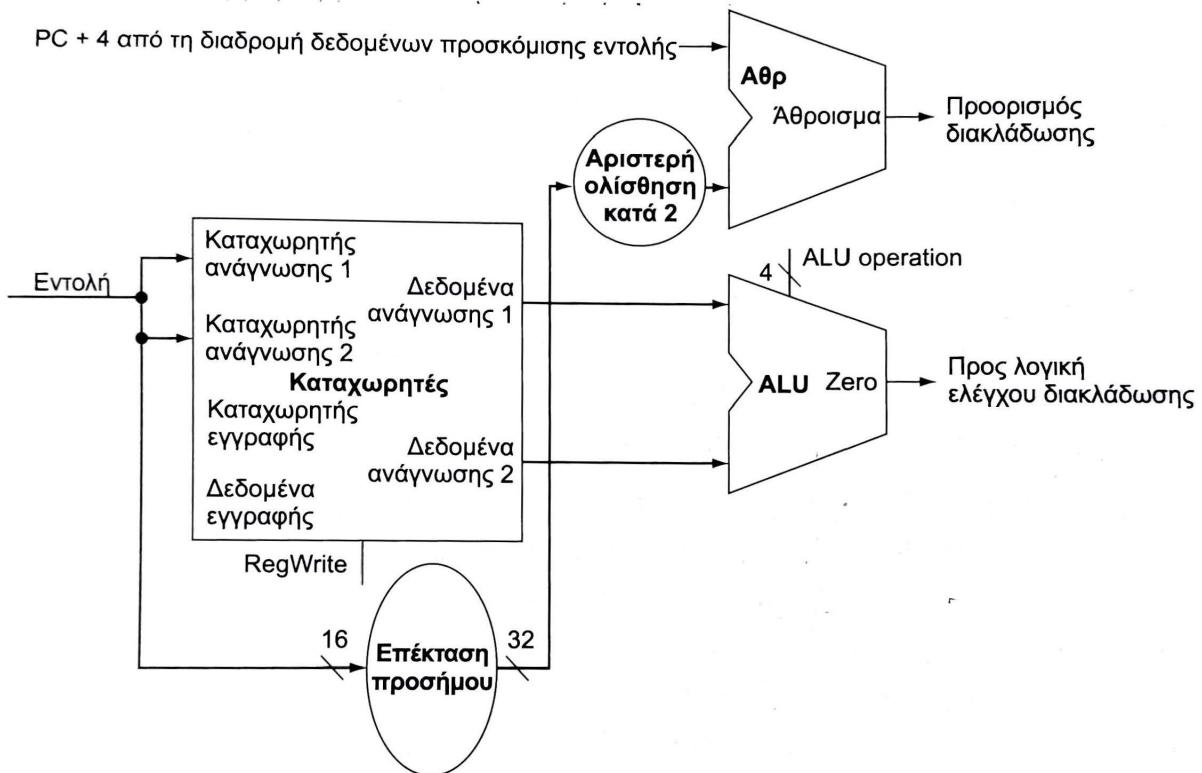
κάποια πράξη και μία λέξη γράφουν πίσω στο αρχείο καταχωρητών. Οι εντολές I-format χρησιμοποιούν το πολύ δύο καταχωρητές, ενώ οι J-format κανέναν. Οπότε, στο αρχείο καταχωρητών μας θέλουμε δύο εισόδους καθεμία από τις οποίες θα προσδιορίζει τον αριθμό του καταχωρητή από τον οποίο θα γίνει ανάγνωση και μία έξοδο που θα είναι η τιμή που έχει διαβαστεί από τους καταχωρητές. Για να γράψουμε μία λέξη στο αρχείο καταχωρητών θα χρειαστούμε μία είσοδο η οποία θα προσδιορίζει τον καταχωρητή στον οποίο θα κάνουμε εγγραφή και άλλη μία που θα είναι, φυσικά, τα δεδομένα μας. Επομένως, χρειαζόμαστε τέσσερις εισόδους και 2 εξόδους. Οι είσοδοι αριθμού καταχωρητή έχουν εύρος 5 bit για να προσδιορίσουν κάποιον από τους 32 καταχωρητές ( $2^5 = 32$ ), ενώ η είσοδος δεδομένων και οι δύο έξοδοι έχουν η καθεμία 32 bit μήκος.

Επόμενο στοιχείο του επεξεργαστή και ένα από τα κυριότερα, η Αριθμητική και Λογική Μονάδα (ALU). Εδώ, εκτελούνται οι αριθμητικοί και λογικοί υπολογισμοί που χρειάζεται να κάνει ο επεξεργαστής για να εκτελέσει κάποια εντολή. Οι είσοδοι του είναι 32 bit έκαστη και παράγει μία έξοδο των 32 bit που είναι το αποτέλεσμα του εκάστοτε υπολογισμού, καθώς και μία έξοδο του ενός bit με τιμή 1 αν το αποτέλεσμα είναι 0 ή τιμή ίση με 0 διαφορετικά. Η λειτουργία που θα εκτελεστεί από την ALU εξαρτάται από ένα σήμα ελέγχου, εύρους τεσσάρων bit, το οποίο ονομάζεται σήμα λειτουργίας ALU (ALU operation).

Στη συνέχεια, θα χρειαστούμε μία μονάδα μνήμης για τη δυνατότητα υλοποίησης εντολών μεταφοράς δεδομένων από/προς την μνήμη. Οι εντολές αυτές υπολογίζουν μια διεύθυνση μνήμης προσθέτοντας τον καταχωρητή βάσης στο προσημασμένο πεδίο σχετικής απόστασης 16 bit που περιέχεται στην εντολή. Για παράδειγμα, έστω ότι έχουμε την εντολή lw \$t1, offset\_value(\$t2), ο καταχωρητής βάσης, \$t2, προστίθεται με το offset και υπολογίζεται έτσι η ζητούμενη διεύθυνση μνήμης. Έπειτα η τιμή που διαβάζεται από την μνήμη πρέπει να γραφεί στο αρχείο καταχωρητών και συγκεκριμένα στον καταχωρητή \$t1. Αν από την άλλη είχαμε την εντολή αποθήκευσης sw \$t1, offset\_value(\$t2), ο υπολογισμός της διεύθυνσης θα ήταν ίδιος ωστόσο το επόμενο βήμα αλλάζει. Η τιμή που θέλουμε να γράψουμε στην διεύθυνση μνήμης που υπολογίσαμε πρέπει να διαβαστεί από το αρχείο καταχωρητών και συγκεκριμένα από τον \$t1. Στη μνήμη δεδομένων, λοιπόν, θα υπάρχουν μία είσοδος διεύθυνσης, είσοδος για τα δεδομένα που θα γίνουν εγγραφή στην μνήμη, καθώς και σήματα ελέγχου ανάγνωσης και εγγραφής ανάλογα με την λειτουργία που θέλουμε να εκτελέσουμε, φόρτωση ή αποθήκευση. Τα σήματα αυτά είναι αναγκαία για την υλοποίηση καθώς η ανάγνωση

μίας τιμής από μη έγκυρη διεύθυνση μπορεί να προκαλέσει προβλήματα. Επίσης, θα χρειαστούμε και μία μονάδα επέκτασης πρόσημου για το πεδίο σχετικής απόστασης (offset), εφαρμόζοντας επέκταση πρόσημου σε μία προσημασμένη τιμή 32 bit.

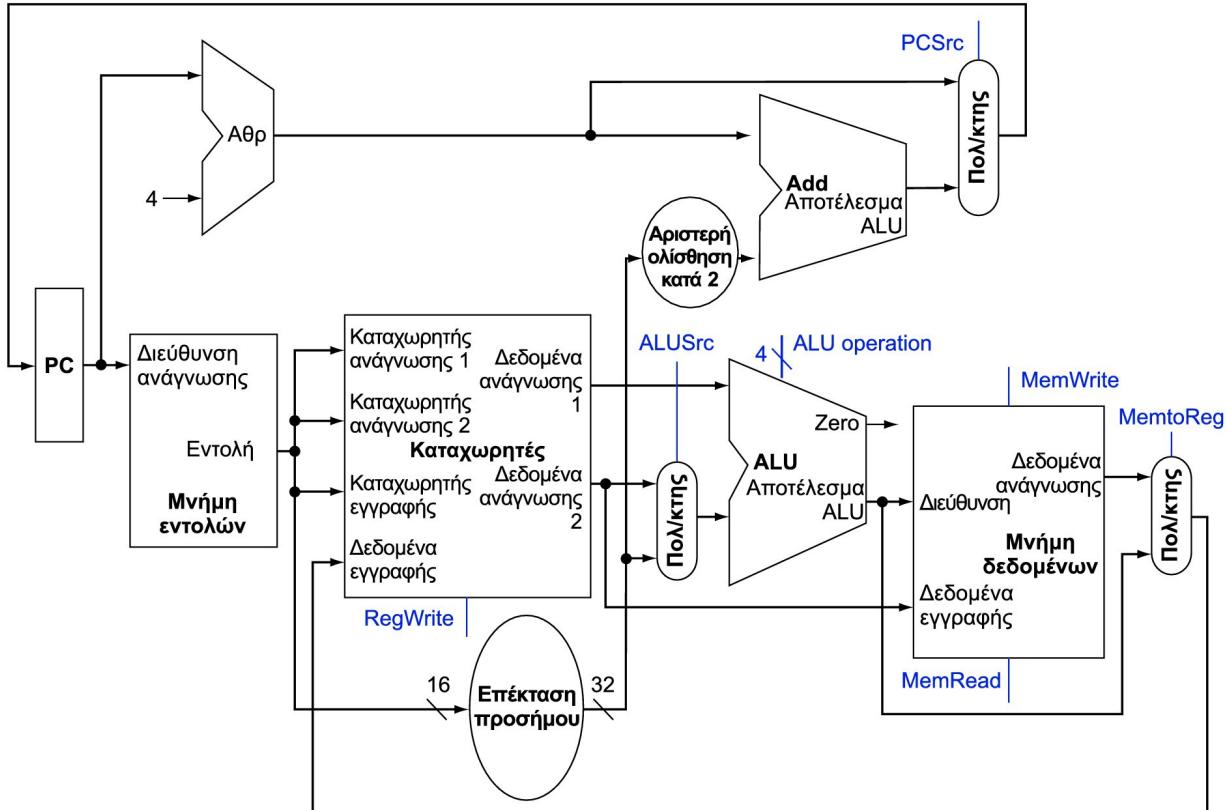
Για την συνέχεια της υλοποίησης θα πάρουμε υπ' όψιν τις εντολές διακλάδωσης. Η εντολή beq είναι της μορφής beq \$t1, \$t2, offset. Έχει, δηλαδή δύο καταχωρητές οι τιμές των οποίων συγκρίνονται για ισότητα και ένα πεδίο σχετικής απόστασης που χρησιμεύει στον υπολογισμό της διεύθυνσης προορισμού διακλάδωσης. Πιο αναλυτικά, για να υπολογιστεί η διεύθυνση προορισμού διακλάδωσης, προστίθεται η σχετική απόσταση (offset), στην οποία έχει εφαρμοστεί επέκταση πρόσημου ώστε να έχει μήκος 32 bit στον καταχωρητή PC, αφού τον έχουμε αυξήσει ώστε να δείχνει την επόμενη εντολή. Ένα πρόβλημα που προκύπτει είναι ότι σύμφωνα με τον ορισμό των εντολών διακλάδωσης, η σχετική απόσταση ολισθαίνει αριστερά κατά 2 bit ώστε να είναι σχετική απόσταση λέξης, αυξάνοντας έτσι το πραγματικό εύρος του πεδίου σχετικής απόστασης κατά έναν συντελεστή 4. Για να αντιμετωπίσουμε αυτό το πρόβλημα χρησιμοποιούμε έναν ολισθητή, ο οποίος ολισθαίνει κατά 2 το πεδίο σχετικής απόστασης. Όταν η συνθήκη είναι αληθής, δηλαδή οι 2 καταχωρητές είναι ίσοι, τότε λέμε πως η διακλάδωση λαμβάνεται (branch taken) και η διεύθυνση προορισμού διακλάδωσης γίνεται η νέα τιμή του καταχωρητή PC. Αν η ισότητα δεν ισχύει, τότε λέμε πως η διακλάδωση δεν λαμβάνεται και ο τρέχον καταχωρητής PC αντικαθίσταται από τον αυξημένο PC. Η σύγκριση αυτή υλοποιείται στην ALU. Στην παρακάτω εικόνα μπορούμε να δούμε τη διαδρομή δεδομένων για μια διακλάδωση, όπως την περιγράψαμε πιο πάνω.



*Εικόνα 2.3: Η διαδρομή δεδομένων για μια διακλάδωση. Αριστερά είναι το αρχείο καταχωρητών, δεξιά η ALU για τον υπολογισμό της συνθήκης και από πάνω είναι ένας αθροιστής που υπολογίζει το άθροισμα του αυξημένου κατά 4 PC και των χαμηλότερων 16 bit της εντολής που έχουν υποστεί επέκταση πρόσημου, στα οποία έχει εφαρμοστεί ολίσθηση.*

Επόμενο βήμα είναι η δημιουργία ενιαίας διαδρομής δεδομένων. Για να το πετύχουμε αυτό αρκεί να συνδυάσουμε κατάλληλα όσα έχουμε υλοποιήσει μέχρι στιγμής. Όπως είδαμε, υπάρχουν διαφορετικές μορφές εντολών που χρησιμοποιούν διαφορετικά τους πόρους της διαδρομής δεδομένων. Για παράδειγμα, οι εντολές τύπου R χρησιμοποιούν 2 καταχωρητές ως εισόδους στην ALU για κάποιον υπολογισμό. Αντίθετα, οι εντολές μνήμης χρησιμοποιούν την ALU για τον υπολογισμό της διεύθυνσης έχοντας σαν είσοδο όμως, έναν καταχωρητή και μία σταθερά (στην οποία έχει εφαρμοστεί επέκταση πρόσημου). Οπότε, για να υλοποιηθεί διαδρομή δεδομένων με μία ALU, πρέπει να υποστηρίξουμε δύο διαφορετικές πηγές για την δεύτερη είσοδο της ALU. Επίσης, πρέπει να υποστηρίξουμε και δύο διαφορετικές πηγές για τα δεδομένα που αποθηκεύονται στο αρχείο καταχωρητή αφού οι τιμές που αποθηκεύονται σε αυτό προέρχονται είτε από την ALU (εντολές τύπου R) ή τη μνήμη (μεταφορά από/προς την μνήμη). Με την τοποθέτηση τριών πολυπλεκτών, έναν στην δεύτερη είσοδο της ALU, έναν στη είσοδο δεδομένων του αρχείου καταχωρητών και έναν για την επιλογή είτε της διεύθυνσης της εντολής που ακολουθεί στη σειρά είτε για την εγγραφή της διεύθυνσης

προορισμού της διακλάδωσης στον PC, το πρόβλημά μας λύνεται.



*Εικόνα 2.4: Η απλή διαδρομή δεδομένων για την αρχιτεκτονική MIPS που υλοποιήθηκε σε αυτό το κεφάλαιο*

### 2.3.3 Έλεγχος

Η διαδρομή δεδομένων μας είναι έτοιμη, ωστόσο, η υλοποίηση μας δε μπορεί να θεωρηθεί ακόμα ως ολοκληρωμένη υλοποίηση του MIPS. Χρειάζεται να προσθέσουμε και μια απλή λειτουργία ελέγχου, η λειτουργία της οποίας είναι αρκετά απλή. Ας αρχίσουμε από τον έλεγχο της ALU. Ανάλογα με την κατηγορία της εντολής, η ALU εκτελεί μία συνάρτηση από αυτές που την έχουμε προγραμματίσει να υπολογίζει (AND, OR, πρόσθεση, αφαίρεση ή set less than). Για τις εντολές μεταφοράς δεδομένων από/προς την μνήμη η ALU χρησιμοποιείται για τον υπολογισμό της διεύθυνσης, δηλαδή εκτελεί μια πρόσθεση. Για την εντολές R, εκτελεί κάποια εκ των πέντε συναρτήσεων που αναφέραμε πριν, ανάλογα με την τιμή του πεδίου funct (τα 6 πιο ασήμαντα bit της εντολής). Για την εντολή διακλάδωσης, η ALU εκτελεί μία αφαίρεση,

ώστε αν είναι μηδέν το αποτέλεσμα, η συνθήκη ισχύει. Μπορούμε να παράγουμε μια είσοδο ελέγχου 4 bit της ALU χρησιμοποιώντας μια μονάδα ελέγχου με εισόδους το πεδίο function της εντολής και ένα πεδίο ελέγχου 2 bit. Το πεδίο αυτό, που ονομάζουμε ALUOp, δείχνει αν η πράξη που θα υλοποιηθεί είναι η πρόσθεση (00) για τις load και store word, αφαίρεση (01) για την beq, ή αν προσδιορίζεται από το πεδίο funct (10). Η έξοδος της μονάδας ελέγχου της ALU είναι ένα σήμα μήκους 4 bit που ελέγχει απευθείας την ALU, παράγοντας έναν από τους 4 συνδυασμούς της παρακάτω εικόνας (Εικόνα 2.5).

Γραμμές ελέγχου ALU	Λειτουργία
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than

Εικόνα 2.5: Οι τιμές που παίρνει η μονάδα ελέγχου της ALU και οι λειτουργίες που αντιστοιχούν σε κάθε μια είσοδο.

Στην Εικόνα 2.6 φαίνεται ο τρόπος που δίνουμε τιμές στις εισόδου ελέγχου της ALU με βάση το ALUOp (2 bit) και το πεδίο funct (6 bit).

Κωδικός λειτουργίας εντολής (opcode)	ALUOp	Λειτουργία εντολής	Πεδίο funct	Επιθυμητή ενέργεια ALU	Είσοδος ελέγχου ALU
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
Tύπου R	10	add	100000	add	0010
Tύπου R	10	subtract	100010	subtract	0110
Tύπου R	10	AND	100100	AND	0000
Tύπου R	10	OR	100101	OR	0001
Tύπου R	10	set on less than	101010	set on less than	0111

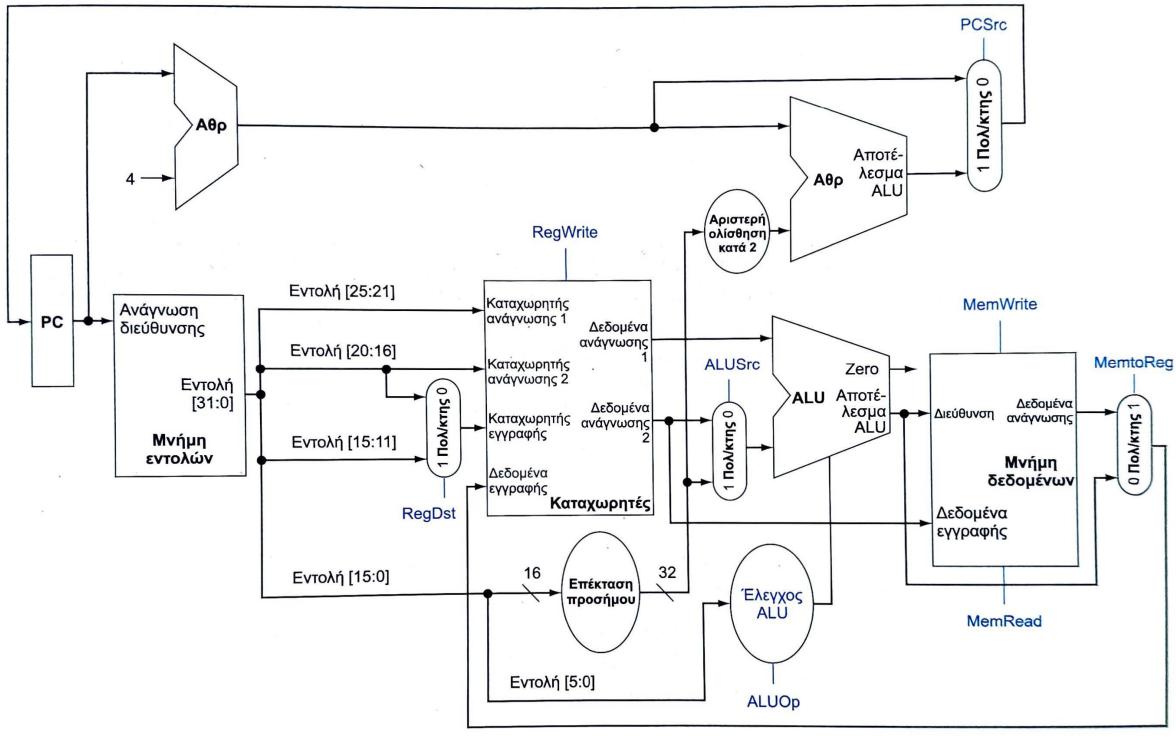
Εικόνα 2.6: Τα αποτελέσματα των σημάτων ελέγχου.

Ο τρόπος ρύθμισης των bit ελέγχου της ALU εξαρτάται από τα bit ελέγχου ALUOp και την κωδικοποίηση του πεδίου funct. Το πεδίο opcode, προσδιορίζει τις τιμές των bit του ALUOp. Όταν ο κωδικός του είναι 00 ή 01 τότε αγνοεί το πεδίο funct διότι το πεδίο αυτό δεν επηρεάζει την ενέργεια της ALU. Αντίθετα, αν είναι 10 τότε πρέπει να εκτελεστεί μια εντολή R, οπότε το πεδίο funct είναι αυτό που προσδιορίζει τι πράξη θα εκτελεστεί, δίνοντας την κατάλληλη είσοδο ελέγχου στην ALU.

Με βάση την μορφή των κατηγοριών εντολών, όπως απεικονίζονται στην Εικόνα 2.1, θα συνεχίσουμε την ανάπτυξη της υπόλοιπης μονάδας ελέγχου. Πρώτα όμως ας κάνουμε κάποιες παρατηρήσεις σχετικές με τις μορφές των εντολών:

- Σε όλες τις κατηγορίες το opcode βρίσκεται στα bit [31:26]. Πλέον, θα αναφερόμαστε σε αυτό ως Op[5:0].
- Οι δύο τελεστέοι προέλευσης προσδιορίζονται από τα πεδία rs[25:21] και rt[20:16] στις εντολές τύπου R, τις εντολές διακλάδωσης και στην store.
- Ο καταχωρητής βάσης για τις εντολές μεταφοράς από/προς την μνήμη είναι πάντα το πεδίο rs[25:21].
- Το πεδίο της σχετικής απόστασης για τις εντολές διακλάδωσης και της μεταφοράς από/προς τη μνήμη είναι πάντα τα 16 πιο ασήμαντα bit [15:0] της εντολής.
- Ο καταχωρητής προορισμού βρίσκεται είτε στις θέσεις [20:16] (rt) στην εντολή load, είτε στις θέσεις [15:11] για εντολή μορφής R.

Από την τελευταία παρατήρηση προκύπτει ότι θα χρειαστούμε έναν πολυπλέκτη στην είσοδο του καταχωρητή εγγραφής που θα προσδιορίζει τον αριθμό του καταχωρητή στον οποίο θα γίνει η εγγραφή. Έπειτα, τοποθετούμε ετικέτες εντολής στην διαδρομή δεδομένων. Κάποιες ως σήματα εγγραφής (RegWrite, MemWrite) και ανάγνωσης (MemRead) και κάποια ως σήματα ελέγχου στους πολυπλέκτες. Οι παραπάνω προσθήκες μαζί με το μπλοκ ελέγχου της ALU απεικονίζονται με μπλε χρώμα στην επόμενη εικόνα.



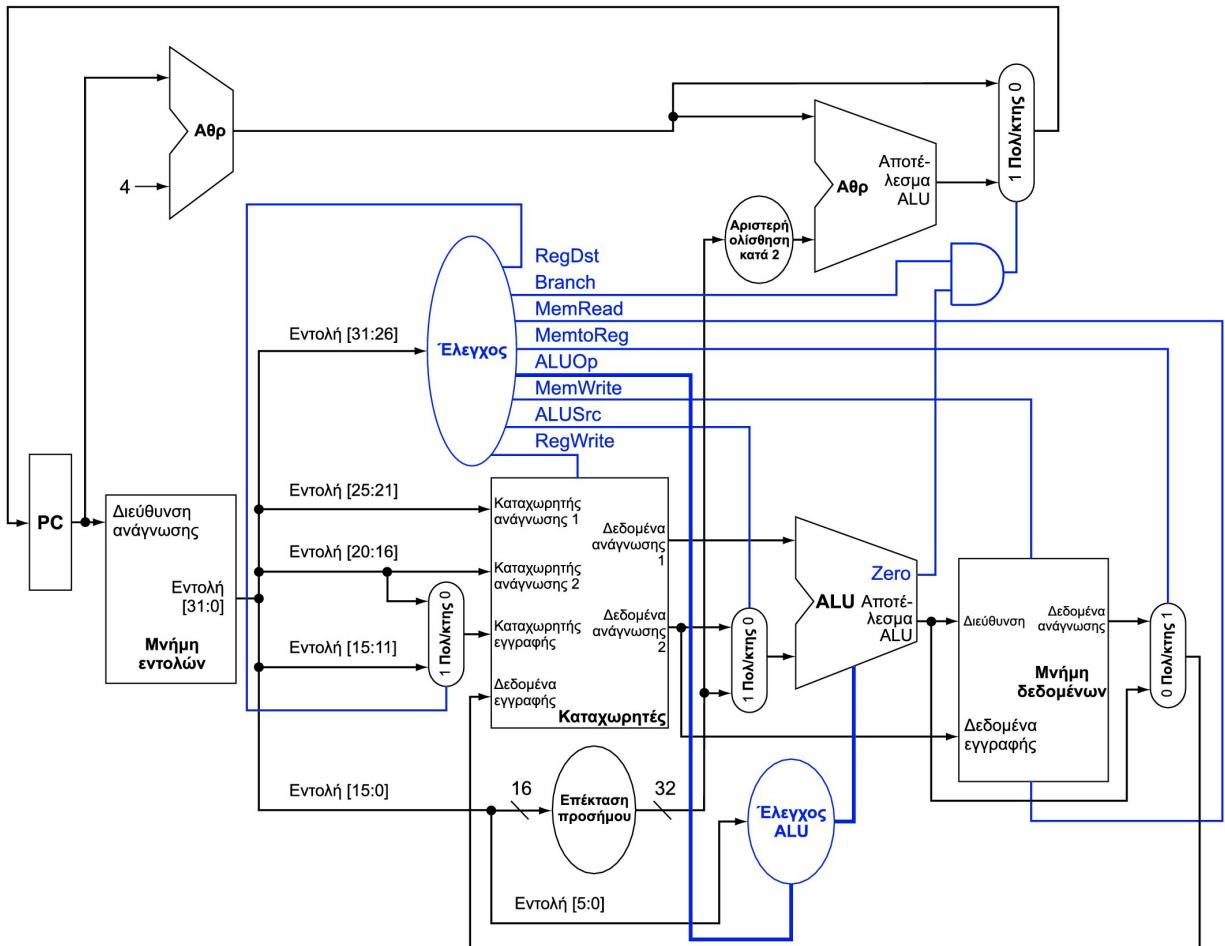
Εικόνα 2.7: Η διαδρομή δεδομένων με όλους τους απαραίτητους πολυπλέκτες και όλες τις γραμμές ελέγχου.

Όλα τα σήματα ελέγχου μπορούν να ρυθμιστούν από την μονάδα ελέγχου με βάση το πεδίο opcode, εκτός από το PCSrc. Για να το ρυθμίσουμε θα δούμε πως πρέπει να δουλεύει. Το σήμα αυτό πρέπει να ισούται με 1 όταν η εντολή που εκτελείται είναι διακλάδωσης και η έξοδος Zero της ALU ισούται και αυτή με 1. Άρα, χρησιμοποιούμε μία πύλη AND, που παίρνει σαν εισόδους το σήμα εξόδου Zero της ALU και ένα σήμα από την μονάδα ελέγχου που ονομάζουμε branch (διακλάδωση) και καταλήγει στον πολυπλέκτη. Ακολουθεί ένας πίνακας με τα αποτελέσματα κάθε σήματος ελέγχου της μονάδας ελέγχου καθώς και μια εικόνα της ολοκληρωμένης διαδρομής δεδομένων με την προσθήκη της μονάδας ελέγχου.



Όνομα σήματος	Αποτέλεσμα όταν είναι απενεργοποιημένο	Αποτέλεσμα όταν είναι ενεργοποιημένο
RegDst	Ο αριθμός καταχωρητή προορισμού για τον Καταχωρητή εγγραφής προέρχεται από το πεδίο rt (bit 20:16).	Ο αριθμός καταχωρητή προορισμού για τον Καταχωρητή εγγραφής προέρχεται από το πεδίο rd (bit 15:11).
RegWrite	Κανένα.	Στον καταχωρητή που καθορίζεται από την είσοδο Καταχωρητή εγγραφής γράφεται η τιμή που περιέχεται στην είσοδο Δεδομένων εγγραφής.
ALUSrc	Ο δεύτερος τελεστέος της ALU προέρχεται από τη δεύτερη έξοδο του αρχείου καταχωρητών (Δεδομένα ανάγνωσης 2).	Ο δεύτερος τελεστέος της ALU είναι τα κατώτερα 16 bit της εντολής αφού υποστούν επέκταση προσήμου.
PCSrc	Ο PC αντικαθίσταται από την έξοδο του αθροιστή που υπολογίζει την τιμή PC + 4.	Ο PC αντικαθίσταται από την έξοδο του αθροιστή που υπολογίζει τον προορισμό τής διακλάδωσης.
MemRead	Κανένα.	Τα περιεχόμενα της μνήμης δεδομένων που καθορίζονται από την είσοδο Διεύθυνσης τοποθετούνται στην έξοδο Δεδομένων ανάγνωσης.
MemWrite	Κανένα.	Τα περιεχόμενα της μνήμης δεδομένων που καθορίζονται από την είσοδο Διεύθυνσης αντικαθίστανται από την τιμή στην είσοδο Δεδομένων εγγραφής.
MemtoReg	Η τιμή που παρέχεται στην είσοδο Δεδομένων εγγραφής των καταχωρητών προέρχεται από την ALU.	Η τιμή που παρέχεται στην είσοδο Δεδομένων εγγραφής των καταχωρητών προέρχεται από τη μνήμη δεδομένων.

Εικόνα 2.8: Τα αποτελέσματα κάθε γραμμής ελέγχου της μονάδας ελέγχου.



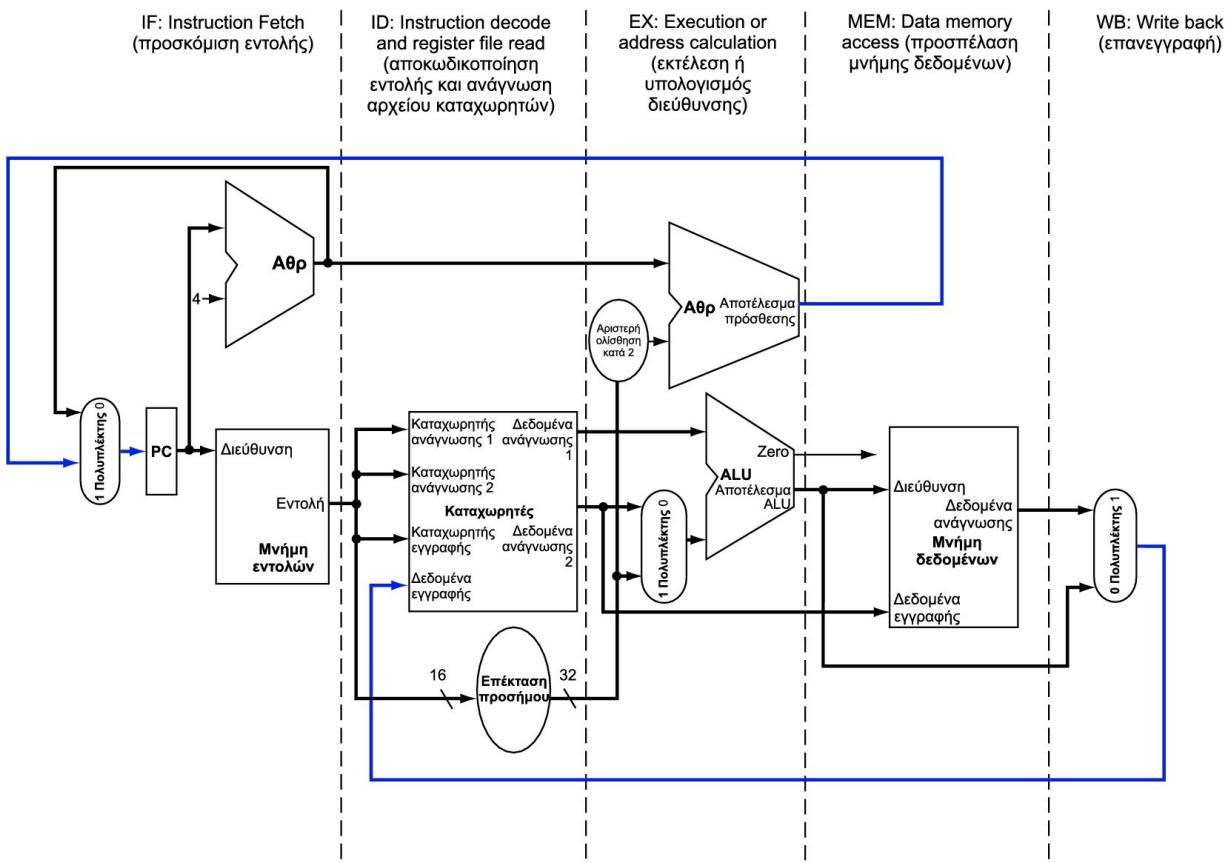
Εικόνα 2.9: Η απλή διαδρομή δεδομένων με την μονάδα ελέγχου

### 2.3.4 Διοχέτευση

Τελευταίο βήμα στην υλοποίηση του επεξεργαστή μας είναι η υλοποίηση διοχέτευσης. Έχουμε αναφερθεί, στα προηγούμενα κεφάλαια, στην σπουδαιότητα και σε κάποια χαρακτηριστικά της τεχνικής της διοχέτευσης. Έχουμε χωρίσει τις εντολές σε πέντε στάδια εκτέλεσης που συνεπάγεται διοχέτευση πέντε σταδίων, δηλαδή σε κάθε κύκλο ρολογιού θα υπάρχει η δυνατότητα εκτέλεσης έως πέντε εντολών. Επομένως, χωρίζουμε την διαδρομή δεδομένων μας σε πέντε μέρη τα οποία αντιστοιχούν σε κάθε στάδιο εκτέλεσης μιας εντολής:

1. IF: Instruction fetch,
2. ID: Instruction decode and register file read,
3. EX: Execution or address calculation,
4. MEM: Data memory access,

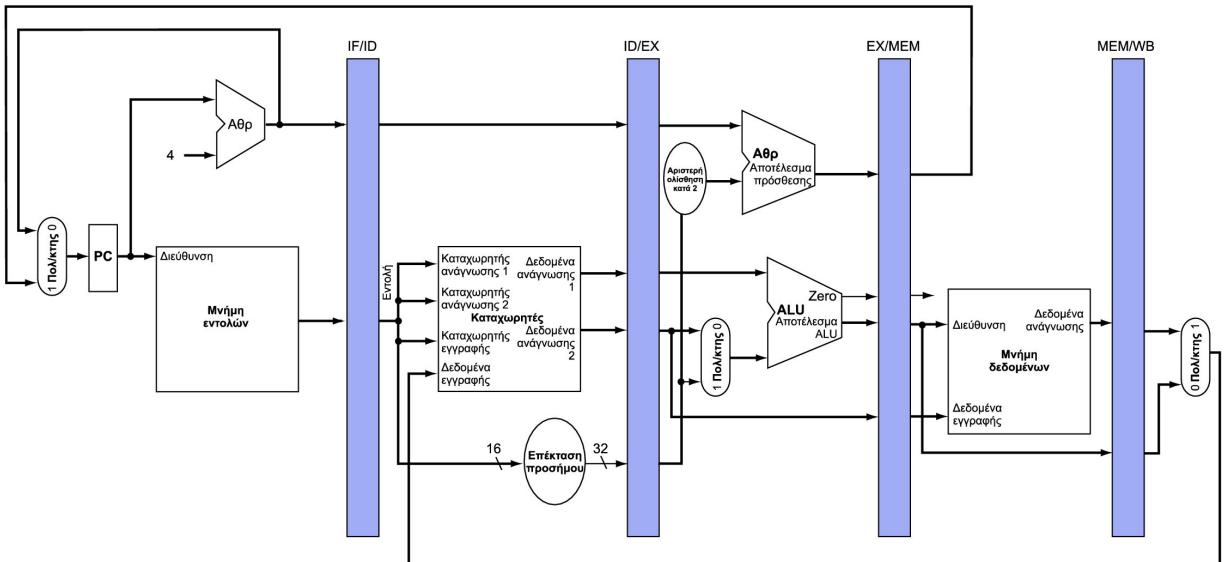
## 5. WB: Write back.



Εικόνα 2.10: Η διαδρομή δεδομένων ενός κύκλου χωρισμένη στα 5 στάδια.

Οι εντολές κατά τα δεδομένα μετακινούνται γενικά από αριστερά προς τα δεξιά στη διάρκεια των πέντε σταδίων καθώς προχωρούν προς την ολοκλήρωση της εκτέλεσής τους. Υπάρχουν, όμως, δύο εξαιρέσεις. Η πρώτη είναι το στάδιο επανεγγραφής, το οποίο τοποθετεί το αποτέλεσμα πίσω στο αρχείο καταχωρητών. Η δεύτερη εξαίρεση αφορά τον μετρητή προγράμματος και συγκεκριμένα την επιλογή της επόμενης τιμής του. Η τρέχουσα εντολή δεν επηρεάζεται ωστόσο είναι πιθανό να επηρεαστούν επόμενες εντολές. Στην πρώτη περίπτωση οδηγεί σε κινδύνους δεδομένων, ενώ η δεύτερη σε κινδύνους ελέγχου.

Για να μπορούν να χρησιμοποιούνται τμήματα της διαδρομής δεδομένων από κοινού κατά την εκτέλεση εντολών προσθέτουμε καταχωρητές που αποθηκεύουν τα δεδομένα σε κάθε στάδιο.



Εικόνα 2.11: Η διαδρομή δεδομένων με διοχέτευση

Τα χρωματισμένα μέρη της Εικόνας 9 είναι οι καταχωρητές διοχέτευσης. Όλες οι εντολές προχωρούν σε κάθε κύκλο ρολογιού από έναν καταχωρητή διοχέτευσης στον επόμενο. Οι καταχωρητές θα πρέπει να έχουν αρκετό εύρος ώστε να αποθηκεύουν όλα τα δεδομένα που αντιστοιχούν στις γραμμές οι οποίες τους διασχίζουν.

Είναι προφανές ότι λείπει η μονάδα ελέγχου που υλοποιήσαμε πιο πριν στην διαδρομή δεδομένων. Η εφαρμογή της εδώ όμως δεν θα είναι τόσο δύσκολη καθώς ήδη η περισσότερη δουλειά έγινε στην προηγούμενη περίπτωση. Συγκεκριμένα, χρησιμοποιούμε την ίδια λογική ελέγχου της ALU, τη λογική διακλάδωσης, τον πολυπλέκτη του αριθμού καταχωρητή προορισμού και τις γραμμές ελέγχου. Το μόνο που χρειάζεται είναι να ορίσουμε τις τιμές ελέγχου στη διάρκεια κάθε σταδίου της διοχέτευσης. Επειδή κάθε γραμμή ελέγχου αντιστοιχεί σε μία μονάδα που είναι ενεργό σε ένα στάδιο διοχέτευσης, χωρίζουμε τις γραμμές ελέγχου σε πέντε ομάδες σύμφωνα με το στάδιο διοχέτευσης:

1. Προσκόμιση εντολής (instruction fetch): Τα σήματα ελέγχου ανάγνωσης της μνήμης και εγγραφής PC είναι πάντα ενεργοποιημένα, οπότε ο έλεγχος δεν είναι αναγκαίος σε αυτό το στάδιο.
2. Αποκωδικοποίηση εντολής και ανάγνωση αρχείου καταχωρητών

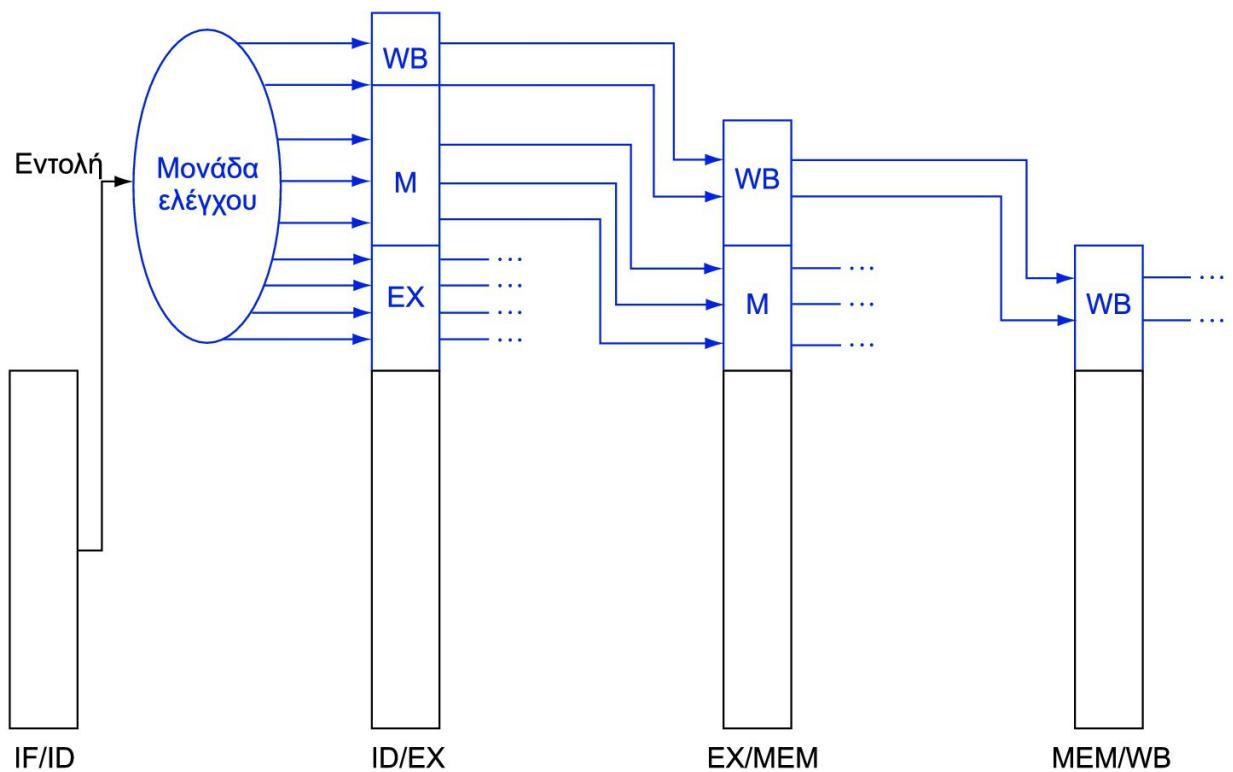
(instruction decode and register file read): Όπως και στο πρώτο, έτσι και σε αυτό το στάδιο συμβαίνει το ίδιο πράγμα σε κάθε κύκλο ρολογιού, οπότε ούτε εδώ υπάρχουν προαιρετικές γραμμές ελέγχου για να πάρουν τιμές.

3. Εκτέλεση και υπολογισμός διεύθυνσης (execute and address calculation): Τα σήματα που πρέπει να πάρουν τιμές είναι τα RegDst, ALUOp και ALUSrc. Τα σήματα επιλέγουν τον καταχωρητή αποτελέσματος, τη λειτουργία της ALU και είτε το πεδίο Δεδομένων ανάγνωσης 2 του αρχείου καταχωρητή είτε ένα άμεσο (immediate) πεδίο αφού αυτό έχει υποστεί επέκταση πρόσημου για να πάει στην ALU.
4. Προσπέλαση μνήμης (memory access): Τα σήματα ελέγχου που παίρνουν τιμές σε αυτό το στάδιο είναι τα Branch, MemRead και MemWrite. Αυτά τα σήματα παίρνουν τιμές από τις εντολές branch equal, load και store, αντίστοιχα. Το σήμα PCSrc επιλέγει την επόμενη διεύθυνση στη σειρά, εκτός αν ενεργοποιηθεί το σήμα Branch και το αποτέλεσμα της ALU είναι μηδέν.
5. Επανεγγραφή (write back): Οι δύο γραμμές ελέγχου είναι οι MemtoReg που αποφασίζει αν θα στείλει το αποτέλεσμα της ALU ή την τιμή της μνήμης στο αρχείο καταχωρητών και η RegWrite που γράφει την επιλεγμένη τιμή.

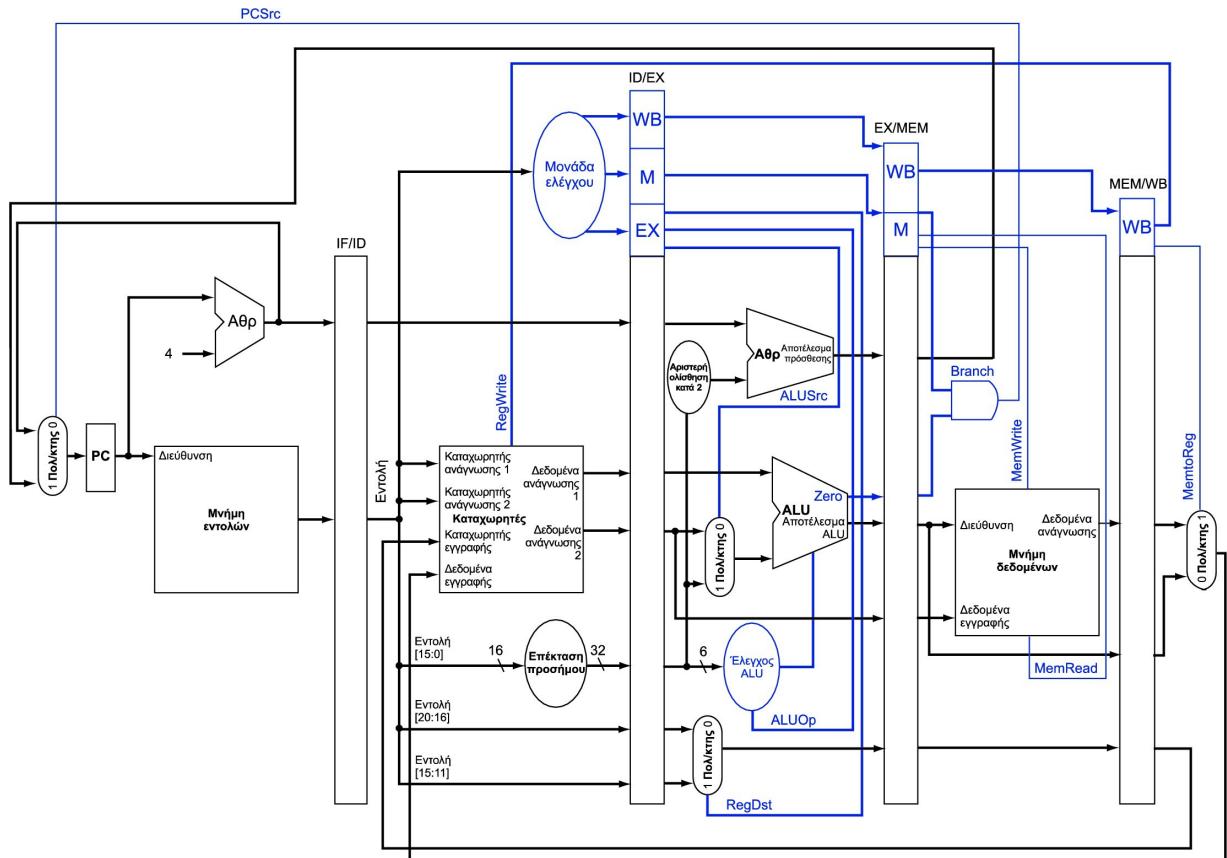
Εντολή	Γραμμές ελέγχου σταδίου Εκτέλεσης/υπολογισμού διεύθυνσης (EX)					Γραμμές ελέγχου σταδίου Προσπέλασης μνήμης (MEM)			Γραμμές ελέγχου σταδίου Επανεγγραφής (WB)	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg	
Μορφή R	1	1	0	0	0	0	0	1	0	
lw	0	0	0	1	0	1	0	1	1	
sw	X	0	0	1	0	0	1	0	X	
beq	X	0	1	0	1	0	0	0	X	

Η τοποθέτηση τιμών στις γραμμές ελέγχου καθορίζεται πλήρως από το πεδίο opcode της εντολής. Η πρώτη γραμμή του πίνακα αντιστοιχεί στις εντολές τύπου R. Για όλες αυτές τις εντολές, τα πεδία καταχωρητών προέλευσης είναι τα rs και rt και το πεδίο

καταχωρητή προορισμού είναι το rd που ορίζει τι τιμές θα πάρουν τα σήματα ALUSrc και RegDst. Επιπλέον, μια εντολή τύπου R γράφει σε έναν καταχωρητή, αλλά ούτε διαβάζει ούτε γράφει στην μνήμη δεδομένων. Όταν το σήμα ελέγχου διακλάδωσης είναι ίσο με 0, ο μετρητής προγράμματος αντικαθίσταται χωρίς συνθίκη, αυξημένος κατά 4, ενώ αν είναι ίση με 1 και το αποτέλεσμα της ALU είναι επίσης 1, τότε ο μετρητής αντικαθίσταται από τον προορισμό της διακλάδωσης. Το πεδίο ALUOp για τις εντολές τύπου R τίθεται ίσο με 10 για να δείξει ότι η είσοδος ελέγχου της ALU πρέπει να παραχθεί από το πεδίο funct. Η δεύτερη και η τρίτη γραμμή αυτού του πίνακα δίνουν τις τιμές των σημάτων ελέγχου για τις lw και sw. Τα πεδία ALUSrc και ALUOp στις περιπτώσεις αυτές παίρνουν τιμές για να εκτελέσουν τον υπολογισμό διεύθυνσης. Τα MemRead και MemWrite παίρνουν τιμές για να εκτελέσουν την προσπέλαση μνήμης. Τέλος, τα RegDst και RegWrite παίρνουν τιμές για μια load ώστε να αποθηκευτεί το αποτέλεσμα στον καταχωρητή rt. Η εντολή διακλάδωσης είναι παρόμοια με μια τύπου R, επειδή στέλνει του καταχωρητές rs και rt στην ALU. Το πεδίο ALUOp για τη διακλάδωση παίρνει τιμή για αφαίρεση (01), που χρησιμοποιείται στον έλεγχο ισότητας. Παρατηρούμε ότι το πεδίο MemtoReg είναι αδιάφορο όταν το σήμα RegWrite είναι μηδέν κι αυτό επειδή δε γίνεται εγγραφή στον καταχωρητή έχει ως αποτέλεσμα η τιμή των δεδομένων στη θύρα δεδομένων εγγραφής καταχωρητή δε χρησιμοποιείται. Έτσι, βάζουμε X στον πίνακα, ώστε να δηλώνει τον αδιάφορο όρο.



Υλοποίηση του ελέγχου σημαίνει να πάρουν οι εννέα γραμμές ελέγχου αυτές τις τιμές σε όλα τα στάδια εκτέλεσης της εντολής. Ο πιο απλός τρόπος είναι να επεκτείνουμε τους καταχωρητές διοχέτευσης, ώστε να περιλαμβάνουν τις πληροφορίες ελέγχου. Αφού, όπως είδαμε πριν, οι γραμμές ελέγχου αρχίζουν από το στάδιο EX μπορούμε να τοποθετήσουμε την μονάδα ελέγχου στο προηγούμενο στάδιο, της αποκωδικοποίησης της εντολής.



Εικόνα 2.12: Η τελική μορφή της υλοποίησής μας.

# Κεφάλαιο 3ο: Σχεδίαση και προσομοίωση του MIPS με τη χρήση του Logisim

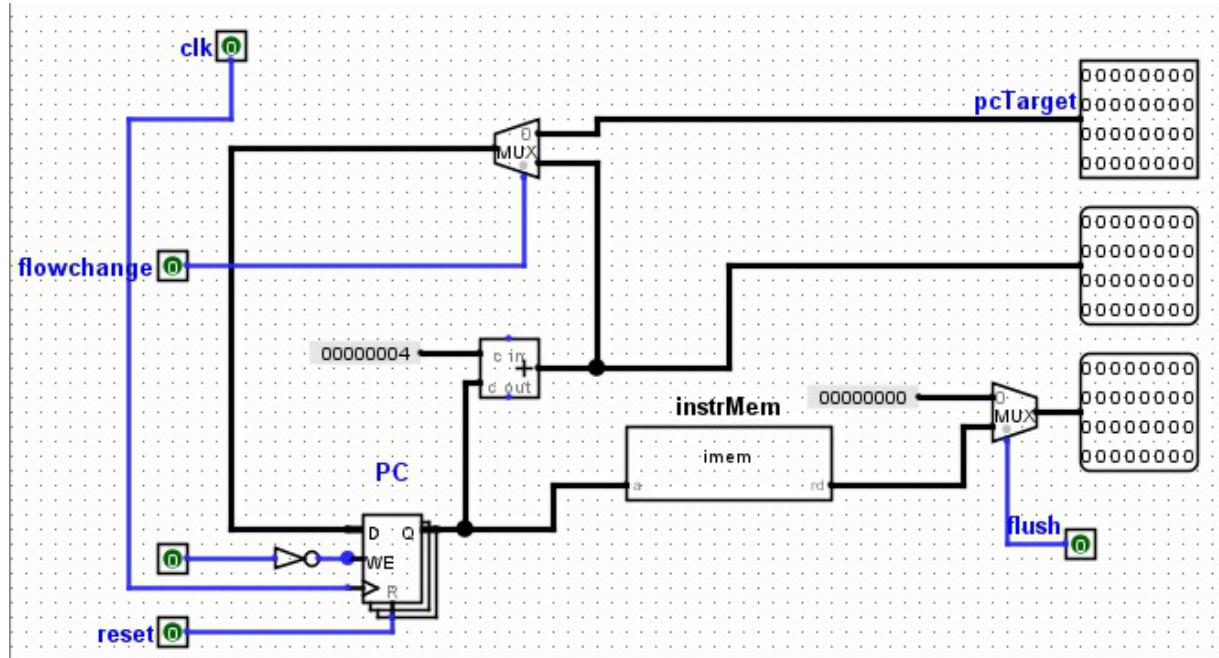
## 3.1 Εισαγωγή

Στο κεφάλαιο αυτό θα περάσουμε από την θεωρία στην πράξη και θα υλοποιήσουμε έναν επεξεργαστή MIPS χρησιμοποιώντας το πρόγραμμα Logisim. Μάλιστα χρησιμοποιούμε μια παραλλαγή του, το Logisim-evolution. Ο επεξεργαστής δεν διαφέρει σε πολλά σημεία με τον επεξεργαστή που περιγράψαμε στο προηγούμενο κεφάλαιο, οπότε σε αυτό το κεφάλαιο θα δώσουμε έμφαση στις διαφορές που υπάρχουν μεταξύ των δύο επεξεργαστών.

Κάθε στάδιο του επεξεργαστή έχει υλοποιηθεί ως ξεχωριστό κύκλωμα στο πρόγραμμα μας. Το ίδιο ισχύει και για τους καταχωρητές διοχέτευσης. Επιπλέον ο επεξεργαστής μας μπορεί να εκτελέσει τις εντολές: jump, lui, ori, addi, addiu.

## 3.2 Σχεδιασμός επεξεργαστή MIPS στο Logisim

### 3.2.1 Στάδιο προσκόμισης εντολής



Εικόνα 3.1: Το στάδιο IF υλοποιημένο στο Logisim

Το στάδιο προσκόμισης εντολής (IF-Instruction Fetch) είναι ίδιο με την υλοποίηση του προηγούμενου κεφαλαίου. Αποτελείται από:

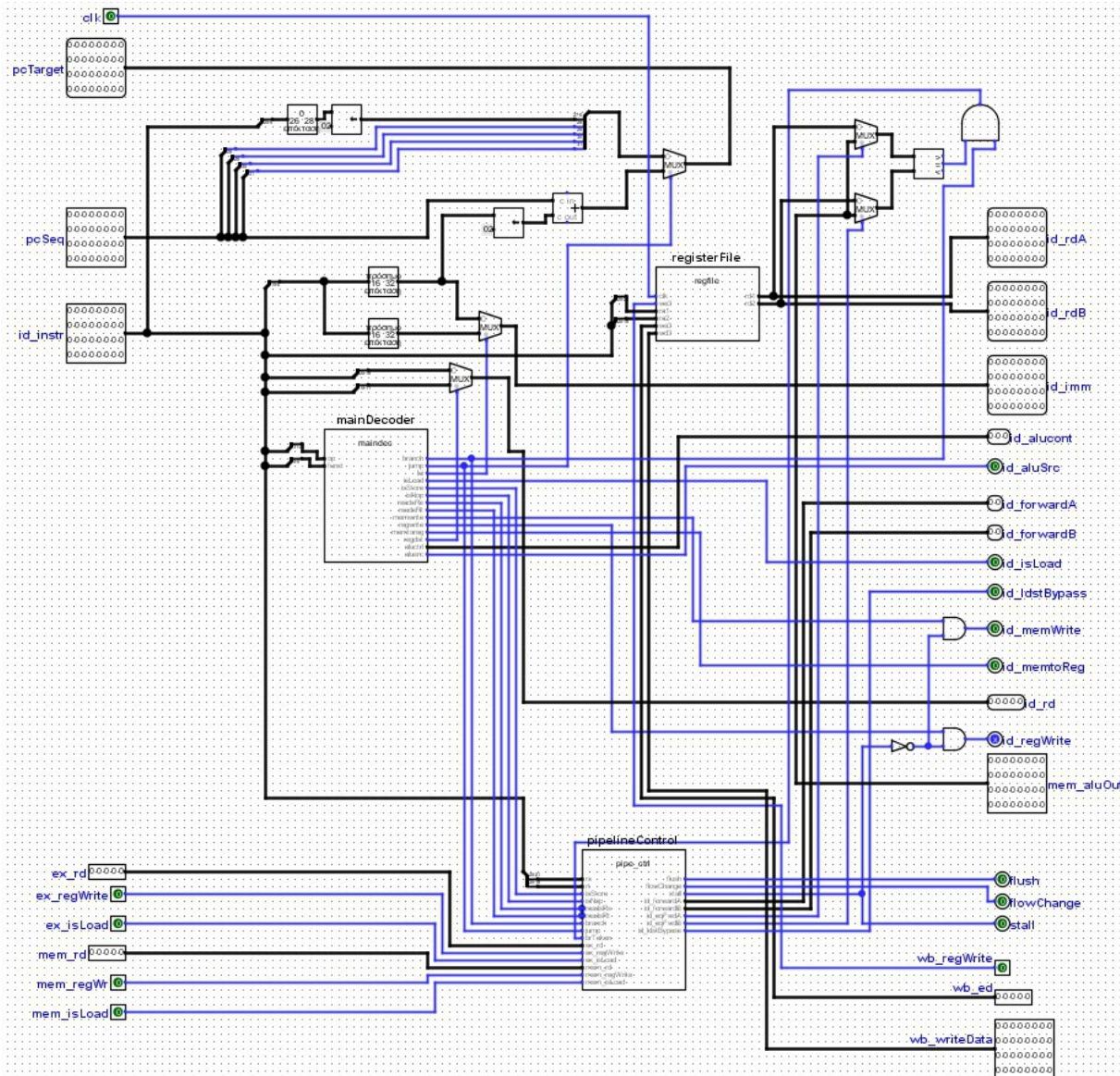
- την μνήμη εντολών (instrMem), που υλοποιείται με χρήση της γλώσσας περιγραφής υλικού VHDL,
- τον μετρητή προγράμματος (PC),
- έναν αθροιστή ώστε να αυξάνει κατά 4 τον μετρητή για να δείχνει την επόμενη εντολή,
- έναν πολυπλέκτη που ελέγχει αν ο μετρητής θα αυξηθεί κατά 4 ή θα πάρει ως είσοδο άλλη διεύθυνση λόγω, για παράδειγμα, μιας εντολής διακλάδωσης.

Logisim: IFstage Στατιστικά					
Στοιχείο	Βιβλιοθήκη	Απλό	Μοναδικό	Επαναλαμβανόμενο	
Ακροδέκτης	Καλωδίωση	8	8	8	
Σταθερά	Καλωδίωση	2	2	2	
NOT Πύλη	Πύλες	1	1	1	
Πολυπλέκτης	Πλέκτες/Κωδικοποίηση	2	2	2	
Αθροιστής	Αριθμητικά	1	1	1	
Καταχωρητής	Μνήμη	1	1	1	
VHDL Entity	HDL-IP	1	1	1	
Επικέτα	Βασική	3	3	3	
ΣΥΝΟΛΟ (δίχως τα υπο-κυκλώματα)		19	19	19	
ΣΥΝΟΛΟ (με υπο-κυκλώματα)		19	19	19	

**Κλείσιμο**

Εικόνα 3.2: Τα στατιστικά του κυκλώματος IFstage.

### 3.2.2 Στάδιο αποκωδικοίησης εντολής & ανάγνωσης καταχωρητών

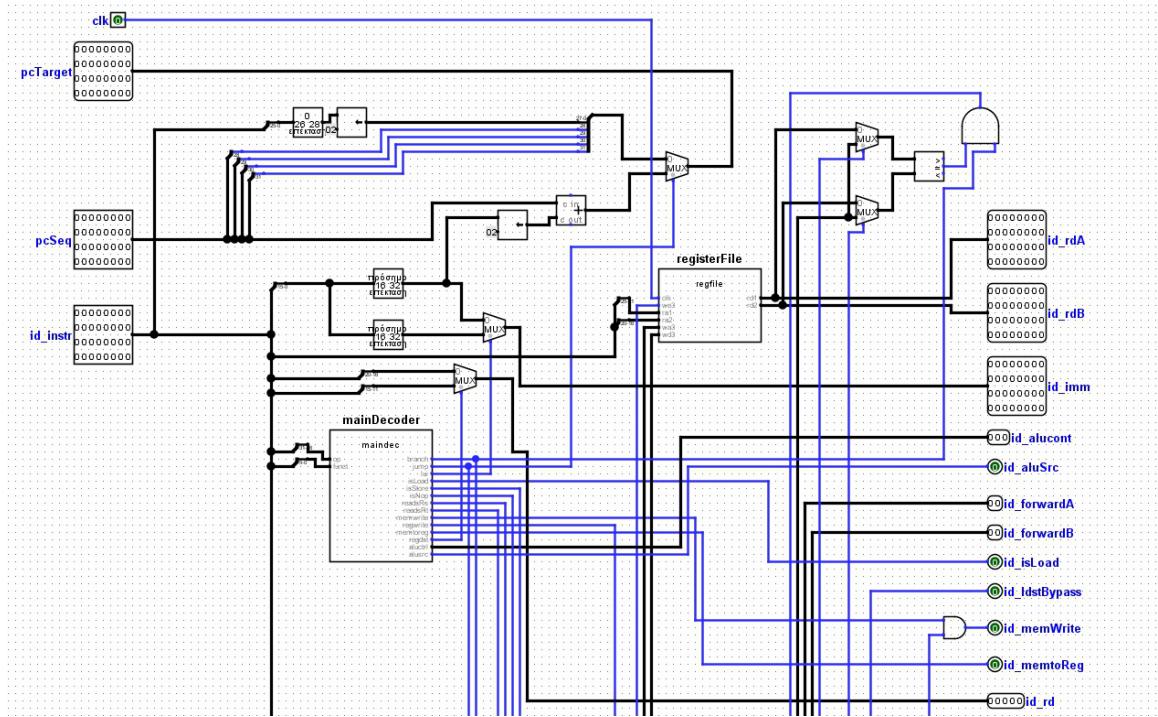


Εικόνα 3.3: Το στάδιο ID υλοποιημένο στο Logisim

Το παρόν στάδιο (ID-Instruction Decode) είναι το πιο περίπλοκο σε αυτόν τον επεξεργαστή, ωστόσο δεν είναι τόσο περίπλοκο όσο μπορεί να φανεί με τη πρώτη ματιά. Αποτελείται από:

- Το αρχείο καταχωρητών, που περιλαμβάνει δύο καταχωρητές εγγραφής και δύο ανάγνωσης και λειτουργεί έτσι όπως περιγράφηκε προηγουμένως.
- Στο αριστερό μέρος της Εικόνας 3.4 βλέπουμε την υλοποίηση της εντολής άλματος (jump) που έχει προστεθεί σε αυτό το κύκλωμα. Με μία επέκταση 2 bit και μία ολίσθηση κατά 2 κάνουμε τα 2 πιο ασήμαντα bit της διεύθυνσης άλματος

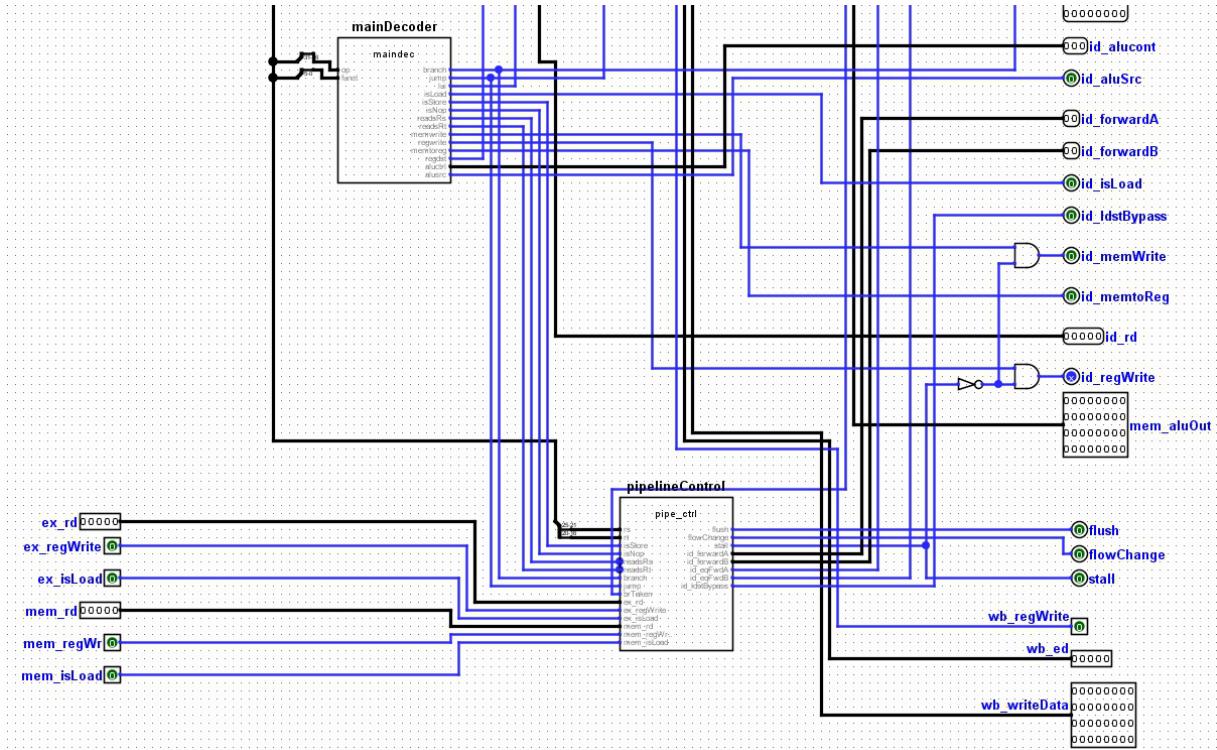
$00_{two}$  [1:0]. Τα επόμενα 26 bit [27:2] προέρχονται από το πεδίο immediate της εντολής, ενώ τα 4 πιο σημαντικά bit [31:28] είναι η τιμή που έχει ο μετρητής προγράμματος κατά την εκτέλεση της εντολής áλματος προσαυξημένος κατά 4. Να σημειωθεί ότι έχει προστεθεί και ο έλεγχος της εντολής áλματος στην μονάδα ελέγχου, mainDecoder, που θα αναλυθεί μετέπειτα.



Εικόνα 3.4: Μέρος του σταδίου ID μεγεθυμένο

- Στο πάνω δεξιά μέρος της Εικόνας 3.4 παρατηρούμε δύο πολυπλέκτες που έχουν από μία είσοδο τις εξόδους του αρχείου καταχωρητών και η δεύτερη είσοδος, η οποία είναι κοινή, είναι το αποτέλεσμα της ALU που μεταφέρεται αριστερά από το στάδιο MEM. Οι έξοδοι των πολυπλεκτών συγκρίνονται και αν είναι ίσες τότε η πύλη AND στέλνει σήμα στην μονάδα ελέγχου της διοχέτευσης, pipelineControl, ότι θα ενεργοποιηθεί η διακλάδωση.
- Η μονάδα ελέγχου, mainDecoder, όπως λέει και το όνομά της, υλοποιείται με έναν αποκαδικοποιητή, ο οποίος επίσης κατασκευάστηκε με χρήση της γλώσσας VHDL και η λειτουργία της είναι όπως ακριβώς περιγράφηκε στο προηγούμενο κεφάλαιο. Ανάλογα με τις τιμές των πεδίων op και funct της εντολής – που αποτελούν ταυτόχρονα και τις δύο εισόδους του αποκαδικοποιητή, ενεργοποιείται η κατάλληλη έξοδος áρα και λειτουργία του επεξεργαστή.
- Τέλος, στο κάτω μέρος του κυκλώματος (και της Εικόνας 5) βρίσκεται ο έλεγχος

διοχέτευσης, επίσης VHDL οντότητα, ένας καταχωρητής που περιλαμβάνει τις πληροφορίες ελέγχου και τις μεταφέρει σε όλα τα στάδια μέσω των καταχωρητών διοχέτευσης που χωρίζουν τα στάδια μεταξύ τους.



Εικόνα 3.5: Το υπόλοιπο μέρος του σταδίου ID

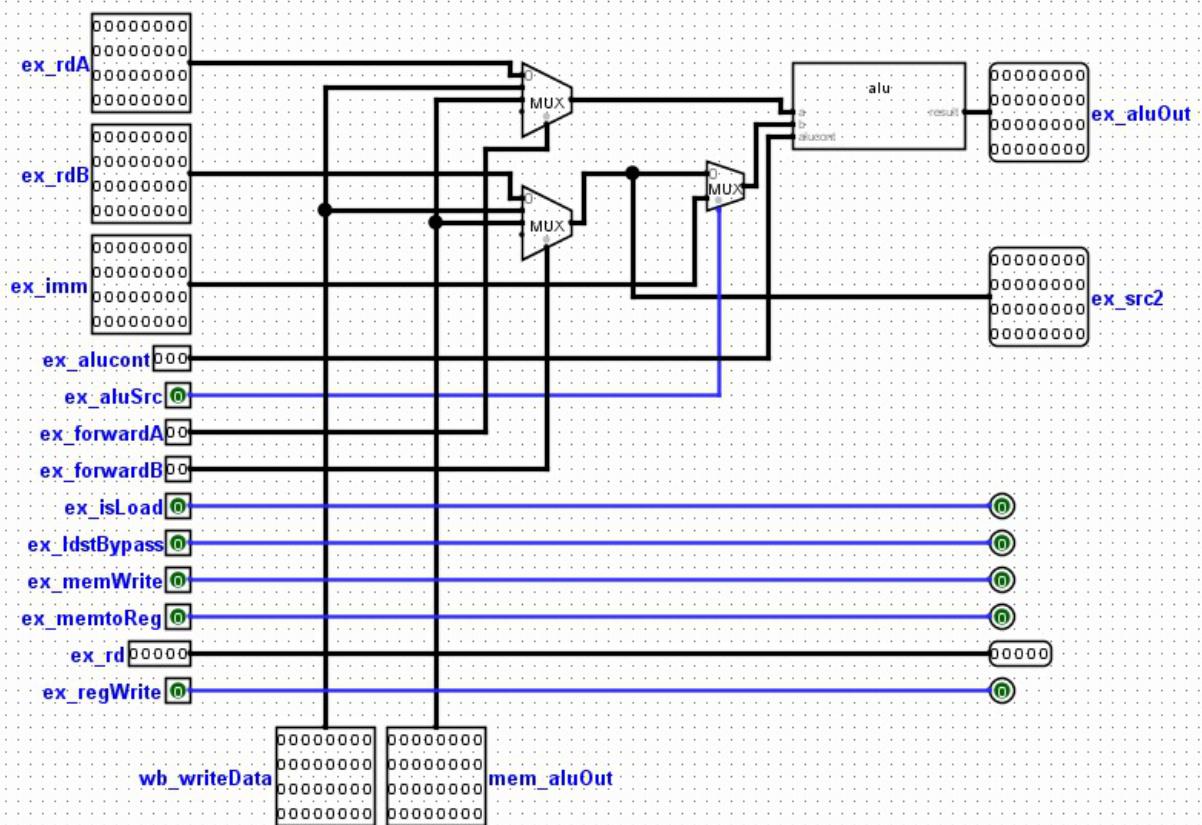
Logisim: IDstage Στατιστικά

Στοιχείο	Βιβλιοθήκη	Απλό	Μοναδι...	Επαναλ...
Διαχωριστής	Καλωδίωση	14	14	14
Ακροδέκτης	Καλωδίωση	30	30	30
Σταθερά	Καλωδίωση	2	2	2
Επέκταση Bit	Καλωδίωση	3	3	3
NOT Πύλη	Πύλες	1	1	1
AND Πύλη	Πύλες	3	3	3
Πολυπλέκτης	Πλέκτες/Κωδικ...	5	5	5
Αθροιστής	Αριθμητικά	1	1	1
Συγκριτής	Αριθμητικά	1	1	1
Ολισθητής	Αριθμητικά	2	2	2
VHDL Entity	HDL-IP	3	3	3
ΣΥΝΟΛΟ (δίχως τα υπο-Κ...		65	65	65
ΣΥΝΟΛΟ (με υπο-Κυκλώμ...		65	65	65

**Κλείσιμο**

Εικόνα 3.6: Τα στατιστικά του κυκλώματος IDstage.

### 3.2.3 Στάδιο εκτέλεσης λειτουργίας ή υπολογισμός διεύθυνσης



Εικόνα 3.7: Το στάδιο EX υλοποιημένο στο Logisim.

Συνεχίζουμε με το στάδιο υπολογισμού, όπως μπορούμε να το πούμε πιο ελεύθερα, καθώς εδώ υπάρχει η ALU που είτε εκτελεί κάποια από τις πράξεις που είναι προγραμματισμένη να εκτελέσει είτε να υπολογίσει μια ζητούμενη διεύθυνση. Όπως φαίνεται από την παραπάνω εικόνα είναι ένα αρκετά απλό στάδιο. Αποτελείται από:

- Την αριθμητική και λογική μονάδα, ή αλλιώς ALU, που είναι υπεύθυνη για την εκτέλεση πράξεων αλλά και τον υπολογισμό πράξεων. Για χάρη ευκολίας έχει υλοποιηθεί κι αυτή ως οντότητα VHDL. Η συγκεκριμένη είναι προγραμματισμένη ώστε να εκτελεί τις λογικές πράξεις AND, OR και ολίσθησης και από αριθμητικές πράξεις την πρόσθεση.
- Δύο πολυπλέκτες 3-σε-1 που παίρνουν ως είσοδο δεδομένα από το στάδιο ID αλλά και από το στάδιο WB και ανάλογα με την τιμή του ex\_forward γίνεται προώθηση ή όχι.
- Έναν πολυπλέκτη 2-1 που διαλέγει ανάμεσα στις εισόδους ex\_rdB ή ex\_imm, δηλαδή μεταξύ καταχωρητή ή σταθεράς ως 2η είσοδο της ALU.

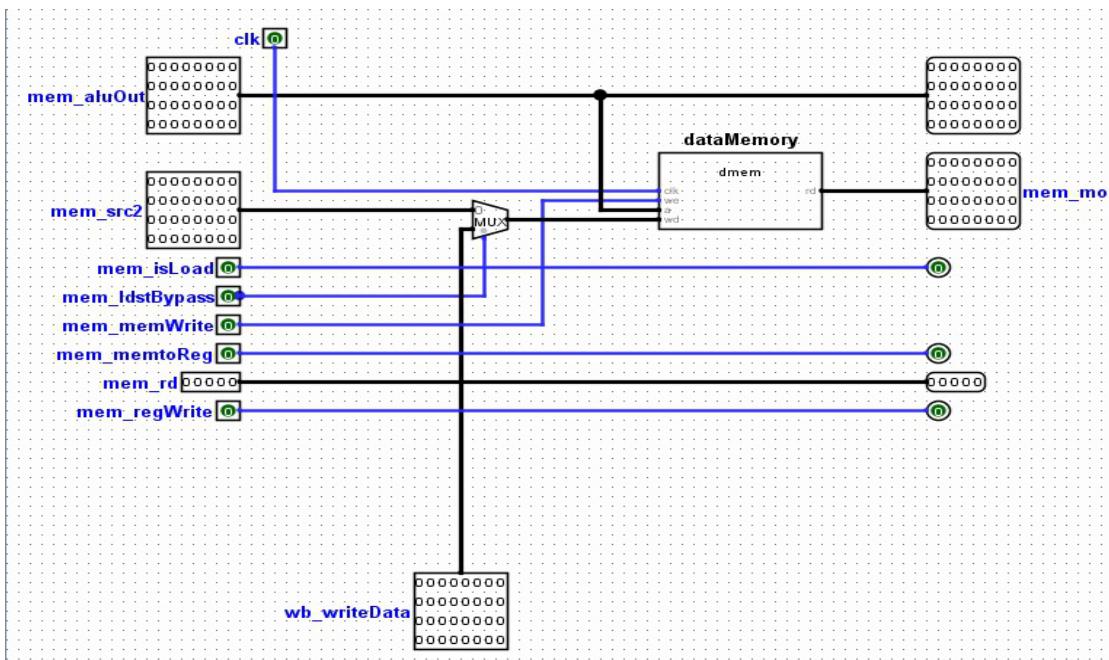
Να σημειωθεί σε αυτό το σημείο ότι η συγκεκριμένη ALU δεν εμφανίζει το σήμα zero όταν η τιμή εξόδου ισούται με μηδέν όπως την περιγράψαμε πριν. Για αυτόν τον λόγο στο προηγούμενο στάδιο συγκρίναμε τις τιμές που παίρναμε ως αποτέλεσμα από την ALU.

Στοιχείο	Βιβλιοθήκη	Απλό	Μοναδι...	Επαναλ...
Ακροδέκτης	Καλωδίωση	23	23	23
Πολυπλέκτης	Πλέκτες/Κωδικ...	3	3	3
VHDL Entity	HDL-IP	1	1	1
ΣΥΝΟΛΟ (δίχως τα υπο-κ...		27	27	27
ΣΥΝΟΛΟ (με υπο-κυκλώμ...		27	27	27

**Κλείσιμο**

Εικόνα 3.8: Τα στατιστικά του κυκλώματος EXstage.

### 3.2.4 Προσπέλαση τελεστέου μνήμης



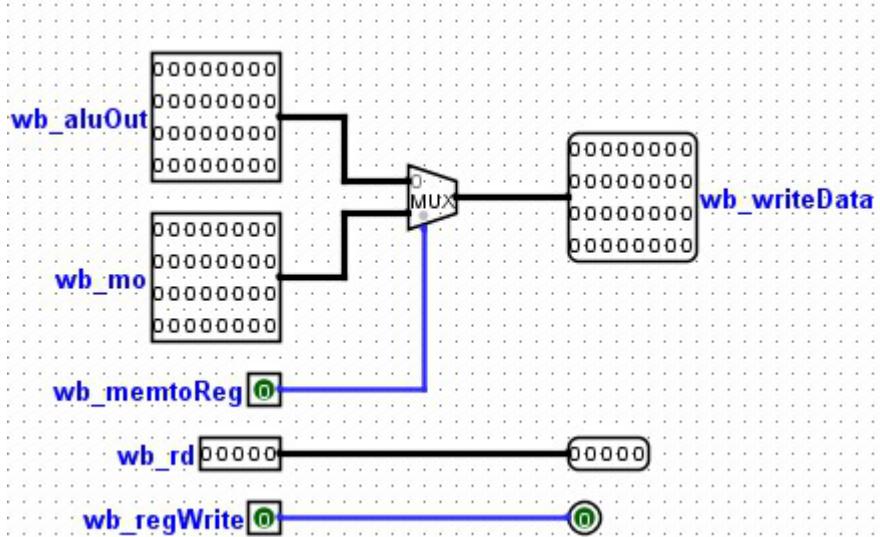
Εικόνα 3.9: Το στάδιο MEM υλοποιημένο στο Logisim.

Και το στάδιο προσπέλασης μνήμης είναι ένα πολύ απλά υλοποιήσιμο στάδιο αφού εκτός της μνήμης δεδομένων που είναι ακόμα ένα στοιχείο υλοποιημένο με χρήση της VHDL είναι η χρήση ενός πολυπλέκτη που καθορίζει αν θα γίνει εγγραφή ή ανάγνωση ανάλογα με την εντολή που εκτελείται.

Logisim: MEMstage Στατιστικά					
Στοιχείο	Βιβλιοθήκη	Απλό	Μοναδι...	Επαναλ...	
Ακροδέκτης	Καλωδίωση	16	16	16	
Πολυπλέκτης	Πλέκτες/Κωδικ...	1	1	1	
VHDL Entity	HDL-IP	1	1	1	
ΣΥΝΟΛΟ (δίχως τα υπο-κ...		18	18	18	
ΣΥΝΟΛΟ (με υπο-κυκλώμ...		18	18	18	
Κλείσιμο					

Εικόνα 3.10: Τα στατιστικά του κυκλώματος MEMstage.

### 3.2.5 Εγγραφή αποτελέσματος



Εικόνα 3.11: Το στάδιο WB υλοποιημένο στο Logisim.

Αποτελεί το τελευταίο και πιο απλό στάδιο. Αποτελείται μόνο από έναν πολυπλέκτη ο οποίος ανάλογα με την εντολή επιστρέφει στο αρχείο καταχωρητών είτε την έξοδο από την ALU είτε δεδομένα από την μνήμη.

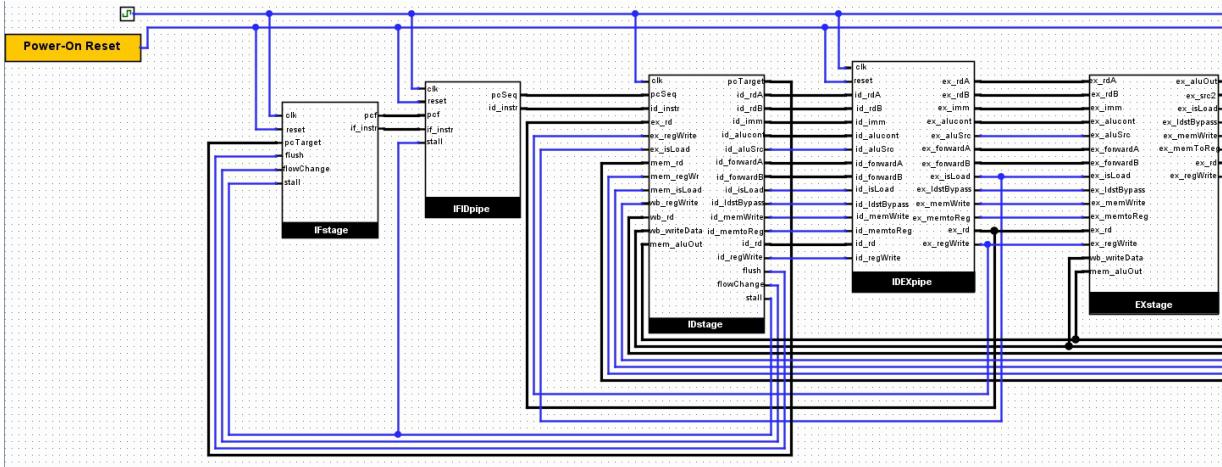
Στοιχείο	Βιβλιοθήκη	Απλό	Μοναδι...	Επαναλ...
Ακροδέκτης	Καλωδίωση	8	8	8
Πολυπλέκτης	Πλέκτες/Κωδικ...	1	1	1
ΣΥΝΟΛΟ (δίχως τα υπο-κ...		9	9	9
ΣΥΝΟΛΟ (με υπο-κυκλώμ...		9	9	9

**Κλείσιμο**

Εικόνα 3.12: Τα στατιστικά του Wbstage.

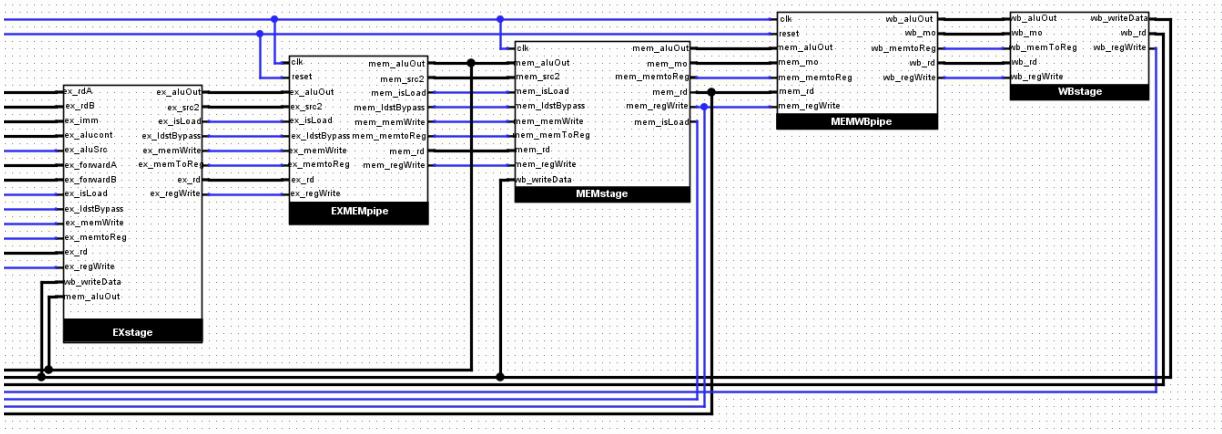
### 3.2.6 Ολοκλήρωση του επεξεργαστή

Είμαστε στο σημείο όπου θα ενώσουμε όλα τα στάδια μεταξύ τους μέσω καταχωρητών διοχέτευσης, ώστε να προκύψει η ολοκλήρωση της ανάπτυξης του επεξεργαστή ως ενιαίο κύκλωμα.



Εικόνα 3.13: Διακρίνονται από αριστερά προς τα δεξιά: το ρολόι, και το κουμπί επανεκκίνησης, το στάδιο IF, ο καταχωρητής διοχέτευσης IF/ID, το στάδιο ID, ο καταχωρητής διοχέτευσης ID/EX και το στάδιο EX.

Το κάθε κύκλωμα εμφανίζεται ως ένα παραλληλόγραμμο, όπου στην αριστερή μεριά του υπάρχουν οι είσοδοι του και στη δεξιά οι έξοδοι. Το Logisim προσφέρει την δυνατότητα στον χρήστη να αλλάξει την εμφάνιση του κυκλώματος εάν το επιθυμεί.



Εικόνα 3.14: Διακρίνονται από αριστερά προς τα δεξιά: το στάδιο EX, ο καταχωρητής διοχέτευσης EX/MEM, το στάδιο MEM, ο καταχωρητής διοχέτευσης MEM/WB και το στάδιο WB.

### 3.3 Προσομοίωση

Μοναδικός στόχος της εργασίας δεν ήταν να δείξουμε πως σχεδιάζουμε και υλοποιούμε έναν επεξεργαστή, αλλά αφού το κάνουμε αυτό να μπορούμε να τρέχουμε προσομοιώσεις. Έτσι εκτός από το να παρατηρούμε τον τρόπο λειτουργίας του βήμα-βήμα μας δίνεται και η δυνατότητα να τον προεκτείνουμε και να τον αναπτύξουμε, είτε ώστε να υποστηρίζει κι άλλες μορφές εντολών, είτε να εκτελούνται εντολές ταυτόχρονα, είτε κάτι άλλο που πιθανόν να μην έχουμε σκεφτεί ακόμα να εφαρμόσουμε.

Αφού τρέχουμε το πρόγραμμα Logisim-evolution στον υπολογιστή μας και έχουμε ανοίξει τον επεξεργαστή (Αρχείο->Άνοιγμα...) επιλέγουμε Προσομοίωση->VHDL simulation enabled, ώστε να γίνουν compile όλα τα στοιχεία VHDL που έχουμε στο κύκλωμά μας. Αν ολοκληρωθεί επιτυχώς το compile τότε στο VHDL simulator log εμφανίζεται το μήνυμα:

*"Environment builded*

# 1.2

# 0

*TCL\_BINDER\_CONNECTED*

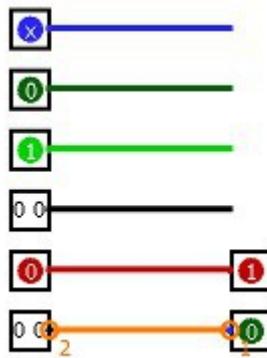
*TCL\_BINDER\_RUNNING"*

Έπειτα επιλέγουμε Προσομοίωση->Αρχικοποίηση Προσομοίωσης ώστε να φορτώσει τις αρχικές τιμές που πρέπει να υπάρχουν στον επεξεργαστή και τέλος Προσομοίωση->Ενεργοποίηση Προσομοίωσης για να ξεκινήσει. Μπορούμε να επιλέξουμε εμείς τον τρόπο λειτουργίας των παλμών, είτε ενεργοποιώντας συνεχόμενους παλμούς και ελέγχοντας την συχνότητά τους, είτε ελέγχοντας τους χειροκίνητα.

Ξεκινώντας το πρόγραμμα πρέπει να έχει ήδη φορτωμένες τις εντολές που θα εκτελέσει στη μνήμη εντολών, imem (αυτό γίνεται αυτόματα με την αρχικοποίηση της προσομοίωσης). Η μνήμη αυτή είναι προγραμματισμένη, αφού είναι στοιχείο VHDL, να φορτώνει τις τιμές που περιέχει το αρχείο "instr\_memfile.txt" που θα πρέπει να περιέχεται στον ίδιο κατάλογο με τον επεξεργαστή την ώρα που αρχικοποιούμε την προσομοίωση. Το αρχείο "instr\_memfile.txt" περιέχει τις εντολές σε κώδικα μηχανής, μεγέθους 32 bit αναπαριστάμενες στο δεκαεξαδικό σύστημα. Για να εξάγουμε από το πρόγραμμά μας τις εντολές σε γλώσσα μηχανής θα πρέπει να χρησιμοποιήσουμε τον MARS. Ο MARS είναι προσομοιωτής προγραμμάτων assembly MIPS και ένα πλήρες γραφικό περιβάλλον ανάπτυξης και εκσφαλμάτωσης προγραμμάτων assembly. Όπως και το Logisim είναι γραμμένος σε Java και τρέχει σε υπολογιστές με οποιοδήποτε λειτουργικό σύστημα.

Ως προς την κατανόηση η προσομοίωση είναι πολύ εύκολη. Σε όλα τα στοιχεία του κυκλώματος, πλην των στοιχείων VHDL, φαίνονται οι τιμές που έχουν κάθε δεδομένη στιγμή και η αλλαγή τους με τον χτύπο του ρολογιού. Οι τιμές στους ακροδέκτες φαίνονται σε πραγματικό χρόνο στο πρόγραμμα. Τα καλώδια

χαρακτηρίζονται από το χρώμα που έχουν:



- **Μπλε:** Το καλώδιο περιέχει τιμή ενός bit αλλά δεν δίνεται κάποια συγκεκριμένη τιμή την παρούσα στιγμή. Καλείται επίσης floating-bit.
- **Σκούρο πράσινο:** Το καλώδιο περιέχει ένα bit με τιμή 0.
- **Ανοιχτό πράσινο:** Το καλώδιο περιέχει ένα bit με τιμή 1.
- **Μαύρο:** Το καλώδιο περιέχει τιμή μεγαλύτερη του ενός bit. Ένα ή περισσότερα bit ενδέχεται να μην προσδιορίζονται.
- **Κόκκινο:** Το καλώδιο περιέχει τιμή σφάλματος. Συχνά προκύπτει όταν μία πύλη δεν μπορεί να καθορίσει την κατάλληλη έξοδο, ίσως επειδή δεν υπάρχουν είσοδοι. Προκύπτει επίσης, όταν 2 εξαρτήματα προσπαθούν να δώσουν διαφορετικές τιμές στο ίδιο καλώδιο όπως στην παραπάνω εικόνα.
- **Πορτοκαλί:** Τα εξαρτήματα που είναι συνδεδεμένα στις άκρες του καλωδίου έχουν διαφορετικό μήκος bit μεταξύ τους, οπότε τα καθιστά ασύμβατα.
- **Γκρι:** Το μήκος bit του καλωδίου είναι άγνωστο και οφείλεται στο γεγονός ότι δεν είναι συνδεδεμένο σε κάποιον ακροδέκτη. Όλες οι είσοδοι και έξοδοι έχουν προκαθορισμένο μήκος bit.

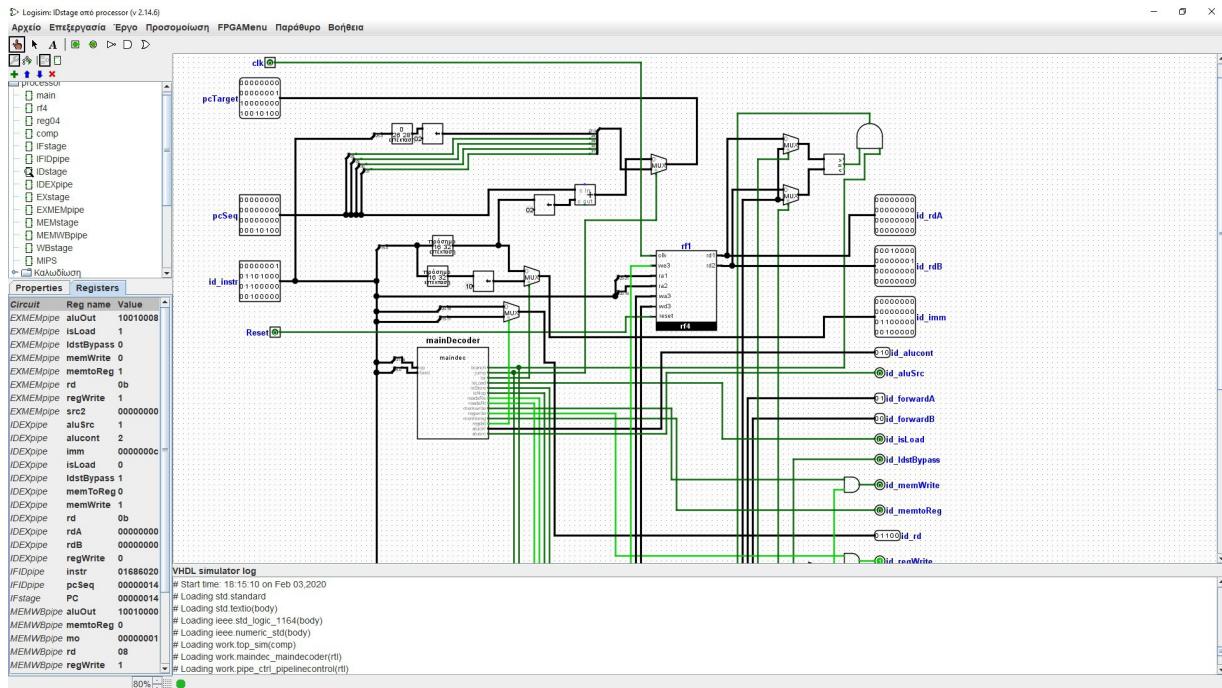
Οι τιμές των καταχωρητών δεν φαίνονται πάνω στο πρόγραμμα την ώρα της εκτέλεσης. Για να μπορέσουμε να δούμε τις τιμές του κάθε καταχωρητή επιλέγουμε και ενεργοποιούμε στις ιδιότητές του την επιλογή "Show in Registers Tab". Έτσι, κατά την

διάρκεια της προσομοίωσης βλέπουμε τι τιμές περνάνε από τους καταχωρητές κάθε στιγμή. Κατά κάποιο τρόπο θυμίζει το waveform που υπάρχει στο Quartus-ModelSim.

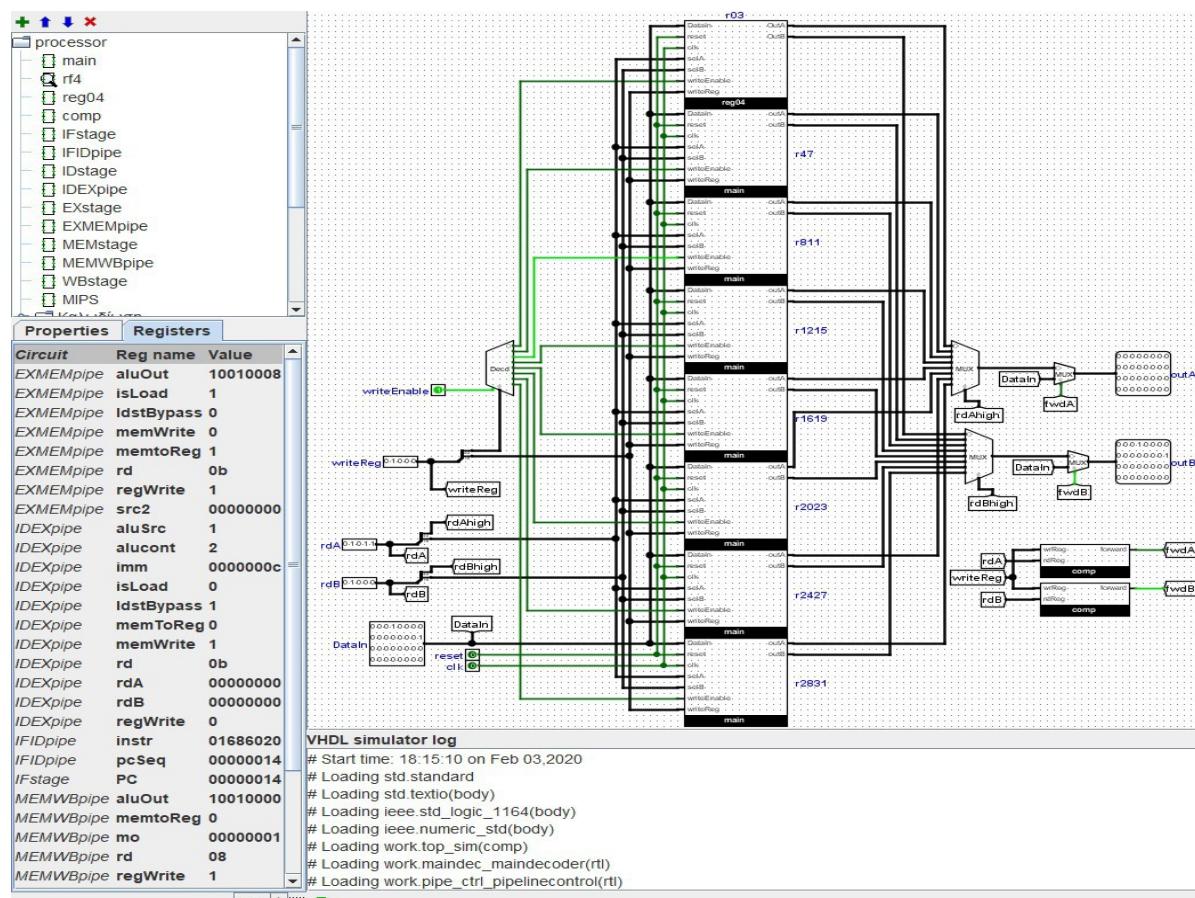
Circuit	Reg name	Value
EXMEMpipe	aluOut	10010008
EXMEMpipe	isLoad	1
EXMEMpipe	IdstBypass	0
EXMEMpipe	memWrite	0
EXMEMpipe	memtoReg	1
EXMEMpipe	rd	0b
EXMEMpipe	regWrite	1
EXMEMpipe	src2	00000000
IDEXpipe	aluSrc	1
IDEXpipe	alucont	2
IDEXpipe	imm	0000000c
IDEXpipe	isLoad	0
IDEXpipe	IdstBypass	1
IDEXpipe	memToReg	0
IDEXpipe	memWrite	1
IDEXpipe	rd	0b
IDEXpipe	rdA	00000000
IDEXpipe	rdB	00000000
IDEXpipe	regWrite	0
IFIDpipe	instr	01686020
IFIDpipe	pcSeq	00000014
IFstage	PC	00000014
MEMWBpipe	aluOut	10010000
MEMWBpipe	memtoReg	0
MEMWBpipe	mo	00000001
MEMWBpipe	rd	08
MEMWBpipe	regWrite	1
main	reg1	00000000

*Εικόνα 3.15: Στιγμιότυπο της Καρτέλας Καταχωρητών (Register's Tab) κατά την διάρκεια εκτέλεσης προσομοίωσης.*

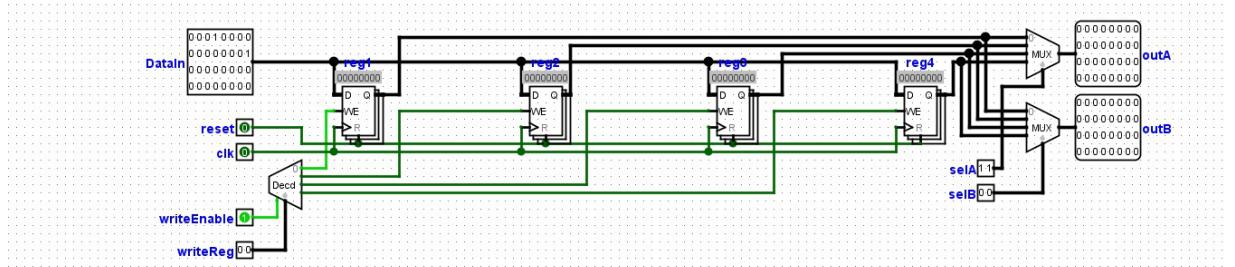
Η προσομοίωση δεν τερματίζει μόνη της, οπότε στην περίπτωση του επεξεργαστή που υπάρχει συγκεκριμένο πλήθος εντολών πρέπει να τις παρακολουθούμε για να ξέρουμε πότε στην ουσία τερματίζει το πρόγραμμα.



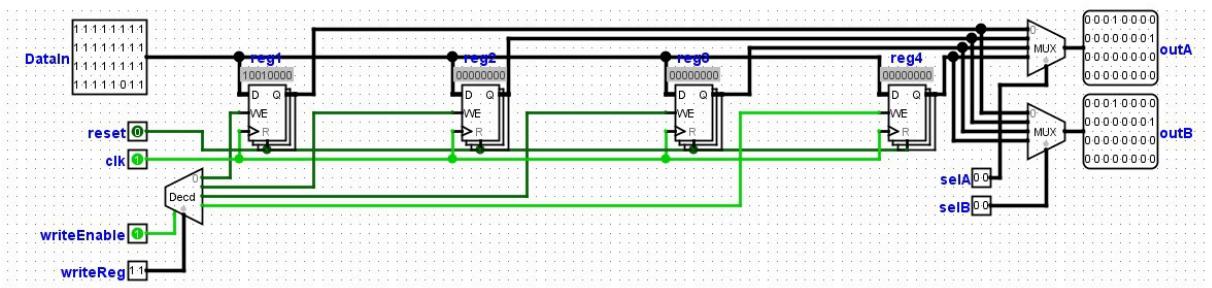
Εικόνα 3.16: Στιγμιότυπο του σταδίου ID κατά την διάρκεια προσομοίωσης.



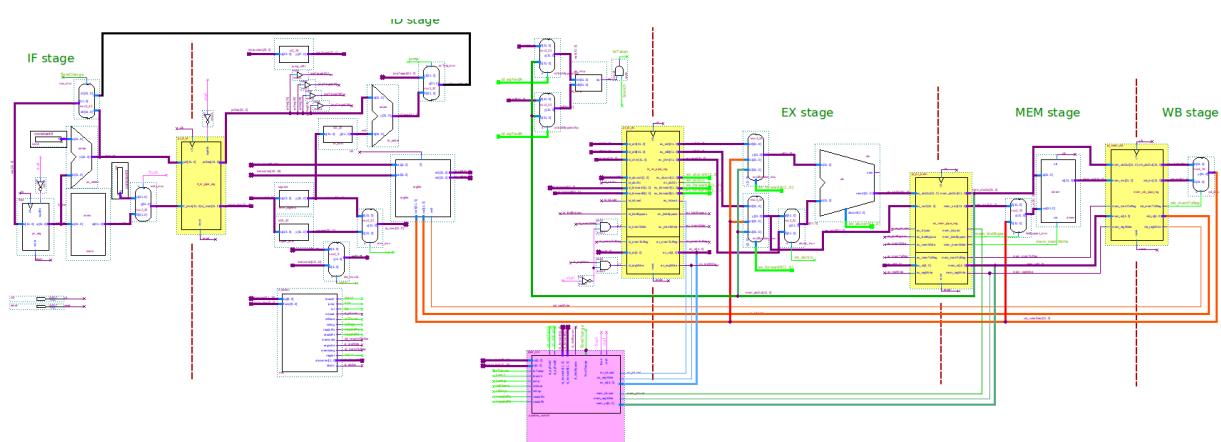
Εικόνα 3.17: Στιγμιότυπο του register file κατά την εκτέλεση της ίδιας εντολής με της Εικόνας 3.16. Η επιλογή writeEnable είναι ενεργοποιημένη και τα 3 πιο σημαντικά bit της μεταβλητής writeReg μας δείχνουν σε ποια τετράδα καταχωρητών θα προωθηθεί η τιμή του Data.



Εικόνα 3.18: Στιγμιότυπο των καταχωρητών 8-11 κατά την εκτέλεση της παραπάνω εντολής. Όπως φαίνεται η τιμή είναι έτοιμη να περάσει στον καταχωρητή 8 (reg8).



Εικόνα 3.19: Στιγμιότυπο των καταχωρητών 8-11 αμέσως μετά το πέρασμα της τιμής στον καταχωρητή 8.



Εικόνα 3.20: Ο επεξεργαστής MIPS υλοποιημένος στο πρόγραμμα Quartus. Όλα τα στοιχεία του κυκλώματος είναι υλοποιημένα με κώδικα σε HDL (Hardware Description Language).

## Κεφάλαιο 4ο: Συμπεράσματα

Στόχος της παρούσας διπλωματικής εργασίας ήταν η σχεδίαση ενός επεξεργαστή MIPS στο πρόγραμμα Logisim και η σύγκριση της σχεδίασης και προσομοίωσης του επεξεργαστή σε σχέση με άλλα προγράμματα, όπως το Quartus-ModelSim της Altera.

To Logisim είναι ένα ελαφρύ και εύκολα επεκτάσιμο πρόγραμμα και υπάρχουν κάποιες παραλλαγές του επίσημου προγράμματος που προστέθηκαν χρήσιμες λειτουργίες. Για παράδειγμα το αρχικό πρόγραμμα δεν περιλάμβανε την προσθήκη εξαρτημάτων VHDL ή chronogram, ωστόσο αυτά προστέθηκαν αργότερα στο Logisim-evolution. Σκοπός του Logisim δεν είναι να αντικαταστήσει τα εμπορικά αυτά προϊόντα αλλά να χρησιμοποιηθεί ως ένα εκπαιδευτικό εργαλείο το οποίο προσφέρει στους φοιτητές και σε όποιον επιθυμεί να εξοικειωθεί με την λειτουργία λογικών κυκλωμάτων, βοήθεια ώστε να κατανοήσουν σε μία πρώτη επαφή την λειτουργία τους, σαν τον επεξεργαστή που υλοποιήσαμε. Αν ο χρήστης έχει βασικές γνώσεις πάνω στα λογικά κυκλώματα τότε δεν υπάρχει και καμία δυσκολία στην κατανόηση λειτουργίας του Logisim που βασίζεται κυρίως σε hardware σχεδιασμό. Προσφέρει ένα απλό και εύκολο στην χρήση περιβάλλον στην σχεδίαση και προσομοίωση που είναι κατανοητή λόγω κυρίως της αμεσότητας της. Ο σχεδιασμός των κυκλωμάτων γίνεται σε ένα γραφικό περιβάλλον παρόμοιο με τα κλασικά προγράμματα σχεδιασμού που συναντάται και σε πολλά άλλα προγράμματα προσομοίωσης. Οι περισσότερες λειτουργίες γίνονται με το ποντίκι του υπολογιστή. Μπορούμε να το χρησιμοποιήσουμε για διαδραστικές ενέργειες στον χώρο εργασίας, όπως επιλογή και μετακίνηση εξαρτημάτων, σύνδεση ή διαγραφή καλωδίων, επαναφορά ιδιοτήτων εξαρτημάτων και άλλα. Επίσης, μετά την ολοκλήρωση του κυκλώματος έχουμε την δυνατότητα να σχεδιάσουμε την μορφή του όπως εμείς επιθυμούμε. Το διαφορετικό, όμως χαρακτηριστικό του Logisim είναι ότι όχι μόνο επιτρέπει την προσομοίωση λογικών κυκλωμάτων, αλλά και την επεξεργασία αυτών κατά τη διάρκεια εκτέλεσης της προσομοίωσης. Ως εκ τούτου, δίνεται η δυνατότητα να διορθώσουμε πιθανά λάθη και σφάλματα χωρίς να χρειάζεται ξεχωριστή επεξεργασία και φόρτωση του κυκλώματος από την αρχή.

Το αρνητικό στην προσομοίωση του Logisim είναι ότι μπορείς να βλέπεις την δεδομένη στιγμή κατάσταση του κυκλώματος και δεν μπορείς να γυρίσεις σε προηγούμενα βήματα ή εντολές. Βέβαια, αυτό θα αποτελούσε χειρότερο χαρακτηριστικό

εάν ο επεξεργαστής μας δεν ήταν υλοποιημένος με διοχέτευση. Με τη διοχέτευση στο Logisim, βλέπουμε από που μεταφέρονται οι τιμές και αν αυτές μεταφέρονται σωστά, δηλαδή αν μηχανισμοί όπως η προώθηση λειτουργούν όπως πρέπει.

Το Quartus-ModelSim αποτελεί ένα επιστημονικό εργαλείο για την σχεδίαση κυκλωμάτων και δοκιμή τους μέσω της προσομοίωσης. Χρησιμοποιείται κυρίως από μεσαίες και μεγάλες επιχειρήσεις και προσφέρουν λύσεις στον σχεδιασμό ηλεκτρονικών αυτόματων συστημάτων, αυτοκινήτων και άλλα. Η σχεδίαση κυκλώματος γίνεται με την χρήση γλωσσών προγραμματισμού περιγραφής υλικού (hardware description languages). Μάλιστα, υποστηρίζει την χρήση πολλών γλωσσών σε ένα κύκλωμα χωρίς να προκαλείται πρόβλημα κατά την μετάφραση (compile). Η προσομοίωση στο ModelSim είναι πολύ πιο σύνθετη και περίπλοκη σε σχέση με το Logisim αλλά είναι πολύ γρήγορη ακόμα και όταν πρόκειται για εξαιρετικά πολύπλοκα προγράμματα. Τα αποτελέσματα της φαίνονται στο κλασικό waveform, έχοντας επιλέξει ποιο σήμα θέλουμε να ελέγχουμε. Φυσικά, από ένα τέτοιο πρόγραμμα δε γίνεται να λείπει η επαλήθευση της λειτουργίας με testbench για περαιτέρω διευκόλυνση και γρηγορότερη επαλήθευση λειτουργίας.

Εν κατακλείδι, το πιο εύκολο πρόγραμμα σε επίπεδο σχεδίασης και προσομοίωσης είναι το Logisim. Είναι σαφώς πιο κατανοητό για κάποιον που δεν έχει εμπειρία και το Logisim του δίνει την δυνατότητα να αποκτήσει. Το ModelSim φυσικά αποτελεί την καλύτερη λύση από άποψη ταχύτητας και δυνατοτήτων που προσφέρει στον χρήστη. Κατά τη γνώμη μου τα δύο προγράμματα δεν μπορούν να συγκριθούν μεταξύ τους, γιατί δεν ανήκουν στην ίδια κατηγορία. Το Logisim είναι ένα εκπαιδευτικό εργαλείο και χρησιμοποιείται για την εκμάθηση των λογικών κυκλωμάτων, ενώ το ModelSim είναι ένα επαγγελματικό και επιστημονικό εργαλείο για προχωρημένους χρήστες.

## Μελλοντική δουλειά

Προκειμένου να υπάρχει μια πιο αντικειμενική άποψη για τα παραπάνω συμπεράσματα θα μπορούσε να μοιραστεί ένα ερωτηματολόγιο μαζί με τον υλοποιημένο επεξεργαστή MIPS σε Logisim και ModelSim σε φοιτητές για δοκιμή των δύο αυτών προγραμμάτων. Από ένα μεγαλύτερο δείγμα θα εξηγάγαμε ασφαλέστερα συμπεράσματα.

Ο ίδιος ο επεξεργαστής έχει κι αυτός περιθώρια βελτίωσης. Μπορεί με την προώθηση και την μονάδα ελέγχου να λύσαμε ζητήματα κινδύνου ελέγχου και να μειώσαμε την καθυστέρηση αλλά δεν εξαλείφθηκαν τελείως. Μια λύση σε αυτό το πρόβλημα είναι η προσθήκη μονάδας ανίχνευσης κινδύνων. Τέλος, η υλοποίηση διοχέτευσης με εξαιρέσεις προσφέρει περαιτέρω μείωση της καθυστέρησης αλλά και αποφυγή κινδύνων που προκύπτουν από τις διακλαδώσεις.

# Βιβλιογραφία

Patterson, David A. και Hennessy, John L. (1994). “Οργάνωση και Σχεδίαση Υπολογιστών: Η Διασύνδεση υλικού και λογισμικού”, Εκδόσεις Κλειδάριθμος, 4η αμερικανική έκδοση.

## Πηγές από το διαδίκτυο

Digital Logic Designer and Simulator, <https://github.com/reds-heig/logisim-evolution>, τελευταία πρόσβαση στις 14/01/2020

RISC Architecture,

<https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/mips>, τελευταία πρόσβαση στις 25/08/2019