Πανεπιστήμιο Κύπρου
Τμήμα Πληροφορικής

Ανάπτυξη Εξομοιωτή Συνόλου Εντολών Αντίστροφης Αρχιτεκτονικής

Συγγραφέας:

Σεργκέι Μπαγκντασάρ

Επιβλέπων καθηγητής:

Pedro Trancoso

30 Μαΐου 2016

Ακαδημαϊκή χρονιά 2015/2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη εξομοιωτή συνόλου
εντολών αντίστροφης αρχιτεκτονικής

Σεργκέι Μπαγκντασάρ

Επιβλέπων Καθηγητής
Pedro Trancoso

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2016

Αφιέρωση

Η παρούσα διπλωματική εργασία είναι αφιερωμένη στον πατέρα μου, ο οποίος έφυγε πρόσφατα από την ζωή, για την δύναμη που άντλησα μέσα από την πίστη που είχε στις ικανότητές μου σε αυτή την δύσκολη περίοδο.

“We can only see a short distance ahead, but we can see plenty there that needs to be done.” — Alan Turing

Ευχαριστίες

Ολοκληρώνοντας την διπλωματική μου εργασία, θα ήθελα να ευχαριστήσω θερμά τον αναπληρωτή καθηγητή του τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου κύριο Pedro Trancoso για την καθοδήγηση και κατανόηση καθ' όλη την διαδικασία εκπόνησης της διπλωματικής εργασίας. Θα ήθελα επίσης να τον ευχαριστήσω για την εμπιστοσύνη που μου έδειξε στην επιλογή του θέματος, καθώς και για τις ιδέες που μου έδωσε για περαιτέρω ενίσχυση της εργασίας αυτής.

Δεν θα μπορούσα να μην ευχαριστήσω τον διδακτορικό φοιτητή του τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου Ανδρέα Διαβαστό, για τις πολύτιμες συμβουλές που μου έδωσε και τον χρόνο που αφιέρωσε για τη βελτίωση, αλλά και την παρουσίαση της παρούσας διπλωματικής εργασίας.

Επίσης, θέλω να ευχαριστήσω τους συμφοιτητές μου από την ερευνητική ομάδα CASPER (Computer Architecture, Systems and Performance Evaluation Research) του τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου για τις εβδομαδιαίες συζητήσεις που είχαμε οι οποίες συνέβαλαν στην αυτοαξιολόγησή μου.

Τέλος, θα ήθελα να πω ένα μεγάλο ευχαριστώ στην οικογένειά μου που με ανέχτηκε και με στήριξε όλα τα χρόνια των σπουδών μου και που μοιράστηκε το πάθος μου για την επιστήμη των ηλεκτρονικών υπολογιστών.

Περίληψη

Λόγω της μεγάλης ανάγκης να μειωθεί η ηλεκτρική ενέργεια που καταναλώνεται από τους ηλεκτρονικούς υπολογιστές καθώς αυτή μετατρέπεται σε θερμότητα η οποία εμποδίζει όλο και περισσότερο την βελτιστοποίηση των ηλεκτρονικών κυκλωμάτων, την τελευταία δεκαετία ερευνάται εντονότερα ο αντίστροφος υπολογισμός -reversible computing-. Τα οφέλη του αντίστροφου υπολογισμού είναι ραγδαία, τόσο υπολογιστικά, όσο και οικονομικά. Υπολογιστικά, η κατακόρυφη μείωση της ηλεκτρικής ενέργειας μπορεί να οδηγήσει σε μεγάλη αύξηση των συχνοτήτων στους επεξεργαστές. Επίσης, σε επίπεδο λογισμικού θα δώσει ευκολότερες και αποτελεσματικότερες λύσεις σε προβλήματα που επανα-υπολογισμού δεδομένων.

Παρόλο που το ερευνητικό ενδιαφέρον συνεχώς αυξάνεται, η έρευνα βρίσκεται ακόμα σε αρχικά στάδια χωρίς να υπάρχουν πολλά εργαλεία ανάπτυξης αντίστροφου υλικού ή λογισμικού. Έτσι, σε αυτή την διπλωματική εργασία έχω αναπτύξει ένα εργαλείο που επιτρέπει την εξομοίωση μιας αντίστροφης συμβολικής γλώσσας προγραμματισμού και το οποίο θα διατίθεται δωρεάν υπό την μορφή του ανοιχτού πηγαίου κώδικα. Το εργαλείο με την ονομασία *“RISA Emulator”* -reversible instruction set architecture emulator- εξομοιώνει την συμπεριφορά του προγράμματος σε έναν αντίστροφο επεξεργαστή, παρέχει περιβάλλον ανάπτυξης προγραμμάτων με ένα σύνολο αντίστροφων εντολών, επιτρέπει την εκτέλεση τέτοιων προγραμμάτων δείχνοντας τις καταστάσεις των καταχωρητών και της μνήμης και τέλος, δίνει την δυνατότητα της αποσφαλμάτωσης με μπροστινή και αντίστροφη κατεύθυνση.

Σε αυτή την εργασία γίνεται μια εισαγωγή στον αντίστροφο υπολογισμό και παρουσιάζεται ο τρόπος προγραμματισμού και χρήσης του προσομοιωτή.

Περιεχόμενα

Σχήματα

Κεφάλαιο 1

Εισαγωγή

1.1 Αντίστροφος Υπολογισμός

Η ιδέα του αντίστροφου υπολογισμού προέρχεται από την επιστήμη της Φυσικής, αφού οι νόμοι της Φυσικής θεωρούνται αυστηρά αντίστροφοι (Toffoli [1]). Σε αυτό προσθέτουμε και την ενέργεια, η οποία εξ ορισμού δε δημιουργείται και δεν καταστρέφεται, αλλά μετατρέπεται σε διαφορετικές μορφές. Αυτό που συμβαίνει στους παραδοσιακούς υπολογιστές είναι ότι μετατρέπουν ένα πολύ μεγάλο ποσοστό της ηλεκτρικής ενέργειας σε θερμική, λόγω της σύνθεσης των κυκλωμάτων που έχουν μια πολλά προς ένα σχέση εισόδων-εξόδων “καταστρέφοντας” έτσι την ηλεκτρική ενέργεια που θα μπορούσε κατά κάποιο τρόπο να ανακυκλώσει. Στην προσπάθεια, λοιπόν, αυτή να διατηρήσουμε την ενέργεια μπορούμε με ένα φυσικό τρόπο να δομήσουμε ηλεκτρονικά κυκλώματα που προσφέρουν σταθερά μικρή -κοντά στο μηδέν- κατανάλωση ηλεκτρικής ενέργειας που δεν είναι ανάλογη του όγκου του υπολογισμού και ταυτόχρονα να αποκτήσουμε τη δυνατότητα εκτέλεσης προγραμμάτων με δύο κατευθύνσεις, κάτι που είναι αδύνατο στους παραδοσιακούς επεξεργαστές.

1.2 Συνεισφορά

Κατά τη φάση της διερεύνησης του αντίστροφου υπολογισμού φανερώθηκαν πολλές ανάγκες για μελλοντική εργασία. Η έρευνα στο συγκεκριμένο πεδίο βρίσκεται ακόμα σε αρχικό στάδιο, αν και την τελευταία δεκαετία έχει επιτευχθεί μεγάλη πρόοδος. Μερικοί τομείς στους οποίους χρειάζεται να εστιαστεί η έρευνα είναι η σύνθεση και βελτιστοποίηση των κυκλωμάτων, ο εμπλουτισμός των απλών αρχιτεκτονικών που έχουν αναπτυχθεί με δυνατότητες σύγχρονων επεξεργαστών και η ανάπτυξη προσομοιωτών -ή εξομοιωτών- αντίστροφων αρχιτεκτονικών και αντίστροφων κυκλωμάτων. Το τελευταίο θεωρώ και το πιο σημαντικό, καθώς αφενός οι προσομοιωτές μπορούν να παρουσιάσουν όλη την δύναμη του αντίστροφου υπολογισμού και αφετέρου, μέσω της προσομοίωσης μπορούν να αναπτυχθούν ικανοποιητικά συστήματα που θα μπορούν να υλοποιηθούν στην πράξη, είτε αυτά είναι υλικό είτε λογισμικό. Έτσι, η δική μου συνεισφορά στην έρευνα του αντίστροφου υπολογισμού είναι ο σχεδιασμός και η ανάπτυξη ενός εξομοιωτή εντολών αντίστροφης αρχιτεκτονικής με γραφικό περιβάλλον. Πιο συγκεκριμένα, η εξομοίωση θα είναι βασισμένη πάνω στην αντίστροφη αρχιτεκτονική “BobIsa Reversible Instruction Set Architecture” [21] η οποία εξετάζεται στο τρίτο κεφάλαιο στα πλαίσια της εισαγωγής στον αντίστροφο προγραμματισμό.

1.3 Κίνητρα

Πέρα από τις μεγάλες προσδοκίες από τον αντίστροφο υπολογισμό -οι οποίες εξετάζονται στο κεφάλαιο δύο-, η επιλογή της υλοποίησης ενός εξομοιωτή ανοιχτού πηγαίου κώδικα με γραφικό περιβάλλον αποσκοπεί στην προσέλκυση νέων ερευνητών στο πεδίο. Θεωρώ πως είναι καίριας σημασίας να προσελκυσθούν νέοι ερευνητές από όλα τα πεδία της επιστήμης των ηλεκτρονικών υπολογιστών σε έναν τομέα με περιορισμένες πηγές πληροφορίσης και δύσκολη πρόσβαση σε εργαλεία. Επίσης, πιστεύω πως κάθε νέος ερευνητής που θέλει να ασχοληθεί με την αντιστρεψιμότητα δεν είναι απαραίτητο να έχει βαθιές γνώσεις αρχιτεκτονικής υπολογιστών και ηλεκτρονικών κυκλωμάτων. Έτσι, θα προσπαθήσω να αναπτύξω έναν εξομοιωτή που να εξελίσσεται με τις εσωτερικές του δομές, ώστε να καλύπτει νέες προσεγγίσεις που αφορούν το υλικό, χωρίς αυτό να επηρεάζει την ευκολία που θα προσφέρει στον τρόπο εξομοίωσης

των αντίστροφων προγραμμάτων. Ακόμη ελπίζω πως με την δυνατότητα ανάπτυξης αντίστροφων προγραμμάτων θα γίνουν πιο ξεκάθαρα τα οφέλη του αντίστροφου προγραμματισμού και θα αυξηθεί το ενδιαφέρον της βιομηχανίας.

1.4 Δομή της διπλωματικής εργασίας

Από αυτό το σημείο και έπειτα, η διπλωματική εργασία θα έχει την ακόλουθη μορφή. Το δεύτερο κεφάλαιο κάνει μια εισαγωγή στον αντίστροφο υπολογισμό μέσα από διάφορους τομείς, έπειτα στο τρίτο κεφάλαιο παρουσιάζονται οι εντολές που υποστηρίζει ο εξομοιωτής. Στο τέταρτο και στο πέμπτο κεφάλαιο παρουσιάζονται το γραφικό περιβάλλον του εξομοιωτή και οι λειτουργίες του, παραδείγματα εκτέλεσης και αποσφαλμάτωσης και τελικά, τεχνικές λεπτομέρειες και υποστηριζόμενα λειτουργικά συστήματα αντίστοιχα. Στο έκτο κεφάλαιο παρουσιάζεται ο ιστότοπος του εργαλείου. Το έβδομο και τελευταίο κεφάλαιο δίνει τους τρόπους που μπορεί κάποιος να εμπλακεί στην περαιτέρω ανάπτυξη του προσομοιωτή και τις ανάγκες για μελλοντική εργασία.

Κεφάλαιο 2

Εισαγωγή στον Αντίστροφο Υπολογισμό

2.1 Προσέγγιση

Κάτω από τον όρο *αντίστροφος υπολογισμός* βρίσκονται διάφοροι τομείς και έννοιες της επιστήμης των ηλεκτρονικών υπολογιστών. Ακολουθώντας μια από κάτω προς τα πάνω προσέγγιση, μπορούμε να περιγράψουμε τον αντίστροφο υπολογισμό βάζοντάς τον κάθε φορά στα πλαίσια του ενός από τα παρακάτω: υλικό του ηλεκτρονικού υπολογιστή, αρχιτεκτονική του υλικού και συνεπώς διάφορες αποφάσεις

σχεδιασμού του υλικού μαζί με το σύνολο των εντολών του και τέλος υψηλού επιπέδου γλώσσες προγραμματισμού. Για να εξηγήσουμε, λοιπόν, περαιτέρω την έννοια της αντιστρεψιμότητας -reversbility- θα πρέπει να την εξηγήσουμε μαζί με το κάθε ένα από τα παραπάνω. Πριν όμως προχωρήσουμε σε αυτό, και για να δώσουμε ενδιαφέρον στη συζήτηση, είναι χρήσιμο να πειστούμε γιατί είναι ώρα να ψάξουμε για νέα μοντέλα υπολογισμού, δηλαδή ποιο είναι το κίνητρο να ασχοληθεί κανείς με την έννοια της αντιστρεψιμότητας.

2.2 Περιοχές εφαρμογής

Είναι γεγονός ότι η τάση που επικρατεί τις τελευταίες δεκαετίες στον τρόπο που κατασκευάζουμε επεξεργαστές έχει συγκεκριμένα πλεονεκτήματα τα οποία η εργασία αυτή δεν αποσκοπεί στο να αναδείξει. Για να υπάρξει, επομένως, στροφή προς έναν εντελώς διαφορετικό τρόπο υπολογισμού θα πρέπει τα οφέλη του να ξεπερνούν κατά πολύ τα όποια αντίστοιχα οφέλη του παραδοσιακού τρόπου υπολογισμού. Στα επόμενα υποκεφάλαια θα προσπαθήσω να αναδείξω μερικά από αυτά που θεωρώ πιο σημαντικά. Επιπλέον περιοχές εφαρμογής υπάρχουν στο σχετικό βιβλίο του Frank [2].

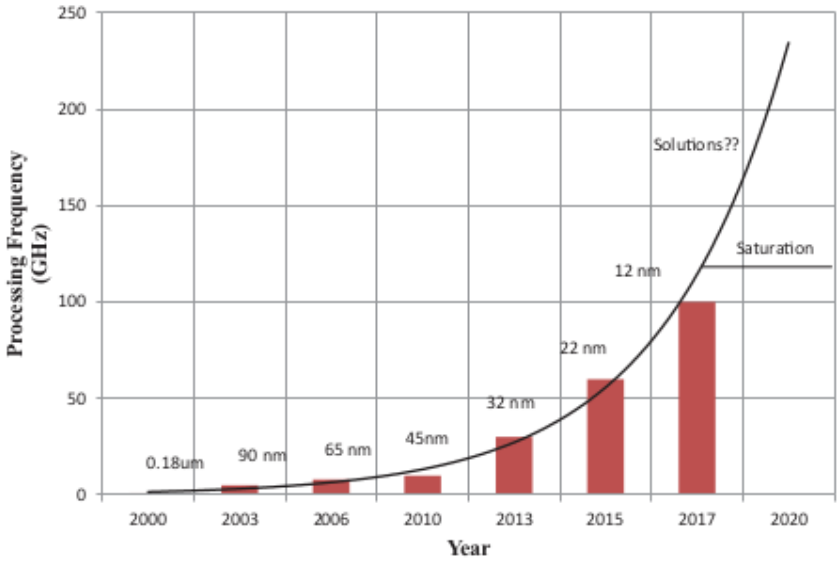
2.2.1 Ενέργεια

Το μεγαλύτερο, ενδεχομένως, κίνητρο στην εξερεύνηση του αντίστροφου υπολογισμού αποτελούσε ανέκαθεν η μείωση κατανάλωσης της ενέργειας η οποία έχει άμεση σχέση με την δύναμη του υπολογισμού. Ποιες είναι, λοιπόν, οι ανησυχίες; Μέχρι και σήμερα, είναι γνωστό ότι η δύναμη των επεξεργαστών μεγάλωνε ραγδαία, κατ' ακρίβεια διπλασιαζόταν περίπου κάθε 18 μήνες σύμφωνα με τον νόμο του Moore. Ο διπλασιασμός αυτός οφείλεται ουσιαστικά στον διπλασιασμό των συχνοτήτων που έγιναν κατορθωτές λόγω της σμίκρυνσης του τρανζίστορ και της συνεχούς βελτίωσης της τεχνολογίας λιθογραφίας, όπως φαίνεται στο Σχήμα 2.1 [3]. Σύμφωνα με την ίδια πηγή, υπάρχουν ισχυρές ενδείξεις ότι με το τέλος της δεκαετίας είναι πιθανό να φτάσουμε στο όριο, αφού το φράγμα των 7nm στην κατασκευή του τρανζίστορ θα έχει επιτευχθεί και δεν μπορεί να ξεπεραστεί, καθώς είναι στο μέγεθος ενός μορίου και με

περισσότερη σμίκρυνση θα παρουσιαστεί ανεξέλεγκτη ροή ηλεκτρονίων. Πέρα όμως από αυτό, δημιουργήθηκαν επιπλέον προβλήματα που έχουν να κάνουν με την εξαγωγή της θερμότητας μέσα από το κύκλωμα, θερμότητα που προκαλείται από την αύξηση του αριθμού των τρανζίστορ και των συχνοτήτων. Ο ρυθμός εξαγωγής της θερμότητας δεν είναι ανάλογος με την αύξησή της, και συνεπώς τα τελευταία χρόνια ο ρυθμός ανάπτυξης των συχνοτήτων μειώθηκε αισθητά.

Αν και γίνεται προσπάθεια μείωσης της ενέργειας χρόνο με τον χρόνο, έχει αποδειχθεί ότι το κάτω φράγμα στην κατανάλωση της ενέργειας από ένα κύκλωμα ορίζεται ως: $E >= k_b T (\ln 2)$ όπου k_b σταθερά Boltzmann (περίπου $1.38\times10-23$ J/K), T η θερμοκρασία του κυκλώματος σε kelvin και ln2 είναι ο φυσικός λογάριθμος του 2 (περίπου 0.69315) [4]. Σύμφωνα με την αρχή του Landauer, υπάρχει κάτω φράγμα στην κατανάλωση της ενέργειας από ένα κύκλωμα, όμως αυτή εφαρμόζεται μόνο στον παραδοσιακό υπολογισμό και όχι στον αντίστροφο. Μόλις πρόσφατα έχει αποδειχθεί και πειραματικά ότι το πιο πάνω όριο δεν είναι κάτω φράγμα για αντίστροφα κυκλώματα. Οι Ren και Semenov [5] κατασκεύασαν κύκλωμα στο οποίο η ενεργειακή κατανάλωση είναι μέχρι και ένα εκατομμύριο φορές λιγότερη από οποιοδήποτε ανάλογο παραδοσιακό κύκλωμα, επιβεβαιώνοντας έτσι την αρχή του Landauer.

Γιατί όμως είναι απαραίτητο να μειώσουμε την ενέργεια; Ακριβώς για να μπορεί να συνεχιστεί με μεγαλύτερο ρυθμό η ανάπτυξη των συχνοτήτων και η εισαγωγή όλο και περισσότερου υλικού μέσα στα κυκλώματα. Θεωρητικά, με τον αντίστροφο υπολογισμό θα μπορούσαμε να πετύχουμε και σχεδόν μηδενική κατανάλωση ενέργειας. Ένας τρόπος να επιτευχθεί αυτό είναι ο υπολογισμός χωρίς διαγραφή περιεχομένου ενός bit κάτι που θα συζητήσουμε αργότερα σε αυτό το κεφάλαιο.



Σχήμα 2.1: Η τάση σμίκρυνσης των τρανζίστορ, ανάπτυξης των συχνοτήτων και το εκτιμώμενο όριο [3]

2.2.2 Ταυτοχρονισμός

Το πεδίο του ταυτοχρονισμού θεωρείται ένα από τα βασικότερα στην προσπάθεια βελτίωσης του υπολογισμού και ειδικά τα τελευταία χρόνια με την ανάπτυξη των πολυπύρηνων και των τεχνικών παράλληλου προγραμματισμού. Η ανάγκη, η οποία προήλθε βέβαια από το πρόβλημα ενέργειας-θερμότητας που αναφέρθηκε στο προηγούμενο υποκεφάλαιο, έφερε έναν νέο τρόπο υπολογισμού που έχει όμως τους δικούς του περιορισμούς. Οι περιορισμοί αυτοί οφείλονται κατά ένα μέρος στο πρόβλημα του συγχρονισμού μεταξύ επεξεργαστών. Ένας τρόπος για να αυξήσουμε τον παραλληλισμό σε εκτελέσεις όπου επεξεργαστές χρειάζεται να συγχρονιστούν είναι να χαλαρώσουμε το μοντέλο εκτέλεσης σε ένα που επιτρέπει στους επεξεργαστές να κάνουν ασύγχρονη εκτέλεση των τοπικών υπολογισμών τους. Αυτό από μόνο του δεν είναι αρκετό, πρέπει ακόμα να δώσουμε στους επεξεργαστές τη δυνατότητα της προς τα εμπρός και προς τα πίσω (αντίστροφης) εκτέλεσης. Δηλαδή, ο κάθε επεξεργαστής θα εκτελεί τους τοπικούς του υπολογισμούς και παράλληλα με αυτό, θα γίνεται έλεγχος για τυχόν παραβιάσεις στις εξαρτήσεις των δεδομένων ανάμεσα στους επεξεργαστές. Αν κάτι τέτοιο εντοπιστεί, τότε με μια προς τα πίσω

εκτέλεση ο επεξεργαστής φτάνει στο σημείο που πρέπει να διορθωθεί και αρχίζει πάλι την προς τα μπρος εκτέλεση. Με τον τρόπο αυτό, είναι δυνατόν να ξεπεραστούν οποιεσδήποτε καθυστερήσεις προκαλούνται από την ανάγκη για συγχρονισμό με μικρό κόστος και χωρίς επιπλέον διαδικασίες στο υλικό για ακύρωση εντολών κ.ο.κ.

Ίδια λογική θα μπορούσε, επίσης, να εφαρμοστεί και σε επίπεδο παραλληλισμού εντολών επεξεργαστή, τη λεγόμενη *speculative execution*. Δηλαδή, είναι ο τρόπος που προσπαθούμε να εκμεταλλευτούμε έναν ενδεχόμενο παραλληλισμό σε επίπεδο εντολών επεξεργαστή εκτελώντας εντολές έξω από την σειρά με την οποία θα εκτελούνταν [6]. Στην προσπάθεια αυτή, γίνεται συνεχής έλεγχος για παραβίαση εξαρτήσεων μεταξύ δεδομένων και εντολών. Εάν τέτοια παραβίαση εντοπιστεί, τότε οι εντολές που χρησιμοποίησαν λάθος δεδομένα ακυρώνονται και εκτελούνται από την αρχή ή τρέχει κάποιος κώδικας διόρθωσης που διορθώνει τα λανθασμένα δεδομένα ώστε να μην ακυρώσει όλο τον υπολογισμό. Για τη διαδικασία αυτή χρειάζονται επιπλέον πόροι, όπως επιπλέον καταχωρητές για να κρατούν προσωρινές τιμές. Αυτή η προσέγγιση μπορεί να αλλάξει αν εκμεταλλευτούμε τις δυνατότητες του αντίστροφου υπολογισμού, δηλαδή αντί να κάνουμε τις πιο πάνω διαδικασίες όταν εντοπιστεί πρόβλημα εξάρτησης απλά να εκτελούμε τις εντολές αντίστροφα μέχρι το προβληματικό σημείο και μετά ξανά προς τα μπρος. Παρόμοιες τεχνικές μπορούν να χρησιμοποιηθούν σε βάσεις δεδομένων (στις συναλλαγές), στην αποσφαλμάτωση κώδικα και σε πολλά συστήματα που προσφέρουν fault tolerance.

2.2.3 Anti-Memoization

Ένα “botlleneck” που παρουσιάζεται στα σημερινά συστήματα είναι αυτό της επικοινωνίας του επεξεργαστή με την κύρια μνήμη. Παρ' όλη την προσπάθεια επίλυσης του συγκεκριμένου προβλήματος με την χρήση της πολυεπίπεδης κρυφής μνήμης, η μνήμη προκαλεί ακόμα μεγάλες καθυστερήσεις, αφού ο χρόνος σε κύκλους που χρειάζεται για να διαβάσεις ή να γράψεις στην μνήμη είναι τάξεις μεγέθους μεγαλύτερος από το χρόνο υπολογισμού. Αυτό που δυσχεραίνει ακόμα περισσότερο την κατάσταση είναι ότι πολλά δεδομένα που είναι στην κρυφή μνήμη και πρόκειται να ξαναχρησιμοποιηθούν διαγράφονται και γράφονται πίσω στην κύρια μνήμη, είτε επειδή

νέα δεδομένα πρόκειται να γραφτούν και δεν υπάρχει άλλος χώρος, είτε από κάποιον αλγόριθμο αντικατάστασης. Αυτή η διαδικασία ονομάζεται *memoization*, με τη χρήση της αντιστρεψιμότητας θα μπορούσαμε να πετύχουμε το *anti-memoization*. Δηλαδή, αντί κάποια δεδομένα να γράφονται πίσω στην μνήμη, να τα αφήσουμε να καταστραφούν γλυτώνοντας έτσι πολλούς κύκλους εγγραφής και μεταγενέστερα διαβάσματος από την μνήμη. Εάν κάποια δεδομένα χρειαστούν ξανά, τότε μπορούμε να κάνουμε αντίστροφη εκτέλεση και να τα ανακτήσουμε. Ένα παράδειγμα τέτοιας προσέγγισης σε επίπεδο καταχωρητών παρουσιάζεται στην έρευνα των Bahi και Eisenbeis [7] .

2.3 Αντιστρεψιμότητα στο υλικό

Το πρώτο ερώτημα που θα έθετε κανείς πριν ασχοληθεί με τον αντίστροφο υπολογισμό είναι αν πράγματι μπορεί να υλοποιηθεί σε επίπεδο υλικού. Η απάντηση είναι πως όχι μόνο μπορεί να υλοποιηθεί, αλλά σχετίζεται απευθείας με τη μείωση της ενεργειακής κατανάλωσης, όπως συζητήθηκε στο υποκεφάλαιο 2.2.1. Παρακάτω, εξετάζουμε την έννοια της συντήρησης της πληροφορίας, της καθολικότητας των πυλών μαζί με κάποιους περιορισμούς και τις πύλες Toffoli και Fredkin σε σχέση με τις πιο πάνω έννοιες. Στη συνέχεια, παρουσιάζεται ένα εργαλείο σύνθεσης και εξομοίωσης αντίστροφων κυκλωμάτων που έχουν μελετηθεί στα πλαίσια της διπλωματικής μου εργασίας, καθώς και μια γλώσσα προγραμματισμού.

2.3.1 Συντήρηση της πληροφορίας

Η συντήρηση της πληροφορίας και κατά συνέπεια της ενέργειας έρχεται από τους νόμους θερμοδυναμικής και τους νόμους συντήρησης της ενέργειας από την επιστήμη της φυσικής. Το κεντρικό σημείο είναι ότι όταν μια πύλη δέχεται ως είσοδο ένα διάνυσμα από bit, τότε ο αριθμός των 1 στο διάνυσμα εξόδου παραμένει ο ίδιος στην πύλη. Αυτή η λογική εξασφαλίζει ότι η ενέργεια σε ένα κύκλωμα δεν καταστρέφεται και δεν δημιουργείται, δίνοντας θεωρητικά δυνατότητα ακόμα και μηδενικής κατανάλωσης από ένα κύκλωμα. Οι Fredkin και Toffoli [8] δίνουν την πύλη

Fredkin ως μια τέτοια πύλη, κάτι που σημαίνει πως ένα κύκλωμα φτιαγμένο αποκλειστικά από πύλες Fredkin έχει την παραπάνω δυνατότητα. Ακόμη, ξέρουμε ότι οι αντίστροφες μηχανές είναι ισοδύναμες με τις μηχανές Turing [9] και άρα, έχουν την ίδια δύναμη υπολογισμού με την πρώτη να έχει το πλεονέκτημα της συντήρησης της πληροφορίας [10].

2.3.2 Καθολικότητα πυλών και περιορισμοί

Μια πύλη ή ένα σύνολο πυλών ονομάζεται καθολικό όταν είναι επαρκές για να συνθέσει οποιοδήποτε λογικό κύκλωμα. Για παράδειγμα, στον παραδοσιακό τρόπο υπολογισμού οι πύλες *“and”*, *“or”*, *“not”* είναι ένα καθολικό σύνολο. Έτσι, προκειμένου μια αντίστροφη πύλη να στηρίζει την έννοια της καθολικότητας, είναι αρκετό να μπορεί να προσομοιώσει τις τρεις πύλες *“and”*, *“or” και “not”*.

Μία αντίστροφη πύλη θα πρέπει να τηρεί περιορισμούς που προκύπτουν από την ίδια την έννοια της αντιστρεψιμότητας. Κατ' αρχάς, η αντιστοιχία της εισόδου με την έξοδο σε μια πύλη θα πρέπει να είναι ένα προς ένα. Αυτό είναι απαραίτητο εφόσον μια είσοδος μπορεί να αντιστοιχεί σε μόνο μια έξοδο και από την άλλη, σε μια αντίστροφη εκτέλεση η ίδια αυτή έξοδος θα πρέπει να αντιστοιχεί στην ίδια εκείνη είσοδο. Με αυτόν τον τρόπο, ένα κύκλωμα που προσθέτει για παράδειγμα 2 bit, κατά την αντίστροφη εκτέλεση να μπορεί να επιστρέψει τους δύο προσθετέους, πιάνοντας ως είσοδο το αποτέλεσμα, καθώς αυτό υποδηλώνει ότι ο αριθμός των εισόδων και των εξόδων είναι ο ίδιος. Ένας δεύτερος περιορισμός έχει να κάνει με το ελάχιστο μέγεθος σε bit που μπορεί να δεχθεί μια πύλη, ο οποίος είναι τουλάχιστον 3. Ο λόγος είναι ότι τα 2 bit δεν παρέχουν αρκετή πληροφορία για τις ανάγκες τις αντιστρεψιμότητας. Για παράδειγμα, η αντιστοιχία για μια πύλη *“and”* είναι η εξής: {(1,1) -> (1,*)} και {{(1,0),(0,1),(0,0)} -> (0,*)} όπου * 1 ή 0. Κατά την αντίστροφη εκτέλεση, αν για παράδειγμα η είσοδος είναι (0,1), δεν είναι εφικτό να γνωρίζουμε από πού προήλθε, αφού θα μπορούσε να παραχθεί από τρεις διαφορετικές εισόδους. Αν όμως είχαμε ένα επιπλέον bit, τότε μια αντιστοιχία (1,0,0)-> (0,0,0),(0,1,0)->(0,1,0) ,(0,0,0)->(0,0,1) θα ήταν επαρκής. Συνοψίζοντας, για μια αντίστροφη πύλη με πλάτος εισόδου i και πλάτος εξόδου u θα πρέπει να ισχύει i=u και u >=3.

2.3.3 Πύλη Fredkin

Η πύλη Fredkin -γνωστή επίσης ως “controlled swap” (CSWAP)- λειτουργεί σαν υπό συνθήκη δρομολογητής, είναι καθολική και συντηρεί την πληροφορία όπως την παρουσιάζουν οι Fredkin και Toffoli [8]. Η μία από τις τρεις εισόδους της πύλης ονομάζεται είσοδος ελέγχου και όταν είναι ίση με 1, τότε οι άλλες δύο είσοδοι περνούν στην έξοδο χωρίς καμιά αλλαγή, αλλιώς ανταλλάζουν τιμές όπως φαίνεται και στον Πίνακα 2.2. Η είσοδος ελέγχου παραμένει πάντα σταθερή στην έξοδο. Ο πίνακας αλήθειας της πύλης φαίνεται στον Πίνακα 2.1 και από αυτόν μπορούμε να εύκολα να καταλάβουμε ότι οι περιορισμοί του ένα προς ένα και του πλάτους των bit τηρούνται, καθώς επίσης ίδιος παραμένει ο αριθμός των 1 και άρα η πληροφορία διατηρείται.

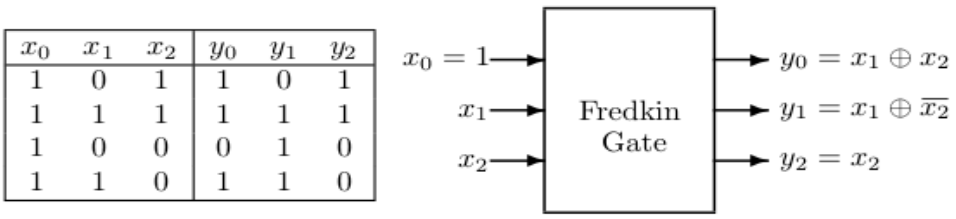
Είσοδοι			Έξοδοι			Μετάθεση
x_0	x_1	x_2	y_0	y_1	y_2	
0	0	1	0	0	1	1 κύκλος
0	1	1	0	1	1	1 κύκλος
1	0	1	1	0	1	1 κύκλος
1	1	1	1	1	1	1 κύκλος
0	0	0	0	0	0	1 κύκλος
0 1	1 0	0 0	1 0	0 1	0 0	2 κυκλοι
1	1	0	1	1	0	1 κύκλος

Πίνακας 2.1: Πίνακας αλήθειας για 3-bit πύλη Fredkin.

Είσοδος	Έξοδος	Περιγραφή
x_0	$y_0=x_2x_0+\neg x_2x_1$	Αν $x_2=1$, τότε $y_0=x_0$ αλλιώς $y_0=x_1$
x_1	$y_1=\neg x_2x_0+x_2x_1$	Αν $x_2=1$, τότε $y_1=x_1$ αλλιώς $y_1=x_0$
x_2	$y_2=x_2$	-

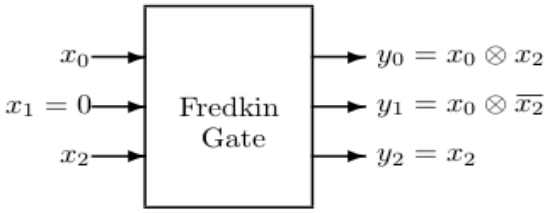
Πίνακας 2.2: Σχέση εισόδου-εξόδου πάνω στην 3-bit πύλη Fredkin

Η καθολικότητα της πύλης μπορεί εύκολα να αποδειχθεί, καθώς με τα σχήματα 2.2, 2.3 και 2.4 που ενσωματώθηκαν από Frank [2] κατασκευάζουμε τις πύλες *“and”*, *“or” και “not”* αντίστοιχα.



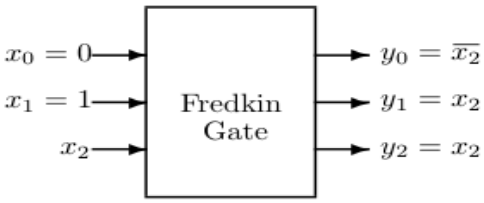
Σχήμα 2.2: Κατασκευή 2-bit πύλης or ⊕ με πύλη Fredkin [2]

x_0	x_1	x_2	y_0	y_1	y_2
0	0	1	0	0	1
1	0	1	1	0	1
0	0	0	0	0	0
1	0	0	0	1	0



Σχήμα 2.3: Κατασκευή 2-bit πύλης and ⊗ με πύλη Fredkin[2]

x_0	x_1	x_2	y_0	y_1	y_2
0	1	1	0	1	1
0	1	0	1	0	0



Σχήμα 2.4: Κατασκευή 1-bit πύλης not με πύλη Fredkin[2]

2.3.4 Πύλη Toffoli

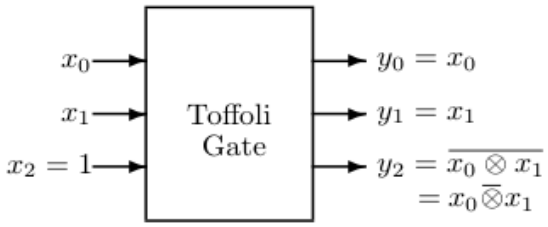
Η πύλη Toffoli -γνωστή και ως “controlled controlled not” (CCNOT)- είναι επίσης αντίστροφη και παρουσιάζεται και αυτή από τους Fredkin και Toffoli [8] . Η διαφορά της από την πύλη Fredkin είναι ότι το πλάτος εισόδου και εξόδου μπορεί να διευρυνθεί μέχρι και n -n μεγαλύτερο ή ίσο με δυο, αν και για n=2 δεν είναι καθολική- χωρίς όμως να κάνει απόλυτη συντήρηση της πληροφορίας. Στον Πίνακα 2.3 βλέπουμε μια 3-bit πύλη όπου η τρίτη έξοδος αλλάζει μόνο όταν οι πρώτες δύο είσοδοι είναι ένα. Από τον ίδιο πίνακα φαίνεται ξεκάθαρα πως η αντιστοιχία είναι ένα προς ένα και το πλάτος είναι επίσης ικανοποιητικό.

Είσοδοι			Έξοδοι			Μετάθεση
x_0	x_1	x_2	y_0	y_1	y_2	
0	0	0	0	0	0	1 κύκλος
0	0	1	0	0	1	1 κύκλος
0	1	0	0	1	0	1 κύκλος
0	1	1	0	1	1	1 κύκλος
1	0	0	1	0	0	1 κύκλος
1	0	1	1	0	1	1 κύκλος
1 1	1 1	0 1	1 1	1 1	1 0	2 κύκλοι

Πίνακας 2.3: Σχέση εισόδου εξόδου πάνω στην 3-bit πύλη Toffoli

Η πύλη Toffoli είναι επίσης καθολική για πλάτος μεγαλύτερο του δύο και αυτό αποδεικνύεται στο Σχήμα 2.5 από M. P. Frank [1] όπου κατασκευάζουμε μια καθολική πύλη *"nand"* χρησιμοποιώντας μια πύλη Toffoli.

x_0	x_1	x_2	y_0	y_1	y_2
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0



Σχήμα 2.5: Κατασκευή 2-bit πύλης nand ⊗ από πύλη Toffoli [2]

2.3.5 Σύνθεση κυκλωμάτων

Το εργαλείο που θα μελετήσουμε είναι το RevKit που παρουσιάζεται από τους Soeken, Frehse, Wille, Drechsler [11]. Γραμμένο στην C++, το RevKit υποστηρίζει τις πύλες Fredkin,Toffoli και Peres και αποτελείται κατά κύριο λόγο από τρία μέρη, όπως αυτά φαίνονται στο Σχήμα 1.6 το οποίο ενσωματώθηκε από την ίδια πηγή. Το πρώτο μέρος είναι ο πυρήνας και παρέχει βασικές συναρτήσεις που μπορούν να

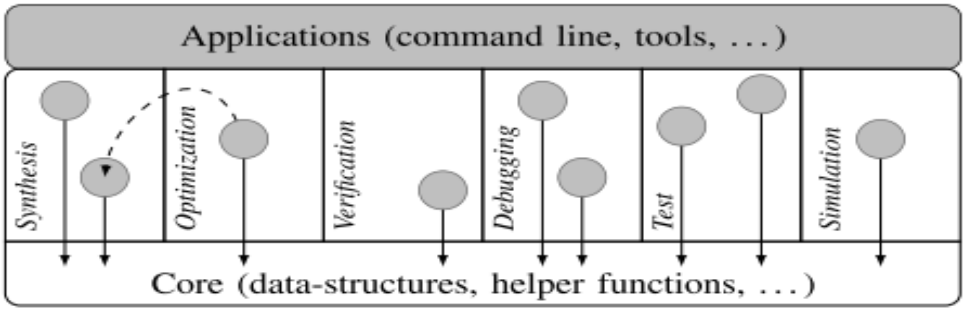
χρησιμοποιηθούν σε κάθε αλγόριθμο όπως συναρτήσεις μέτρησης κόστους, ανάλυσης αρχείων εισόδου βάσει πρότυπων, καθώς και συναρτήσεις που προσφέρουν στοιχειώδεις αλλαγές πάνω στα κυκλώματα.

Το δεύτερο μέρος παρέχει μεθόδους και προσεγγίσεις σύνθεσης κυκλωμάτων και άλλες σχετικές μεθόδους. Πιο συγκεκριμένα, για τη σύνθεση κυκλωμάτων έχουν προταθεί μεθοδολογίες όπως: α) Διάγραμμα Δυαδικής Απόφασης -Binary Decision Diagram (BDD)– που προτάθηκε από Wille και Drechsler [12], β) Αλγόριθμος βασισμένος σε μετασχηματισμούς -A Transformation Based Algorithm- που προτάθηκε από Miller, Maslov και W. Dueck [13], γ) Ιεραρχική σύνθεση που κάνει χρήση της αρνητικής και θετικής αποσύνθεσης Davio -Hierarchical Synthesis of Reversible Circuits Using Positive and Negative Davio Decomposition- που προτάθηκε από Soeken,Wille και Drechsler [14]. Οι μέθοδοι σύνθεσης δεν είναι, όμως, αρκετά αποδοτικές. Μερικές μετρικές της αποδοτικότητας των αντίστροφων κυκλωμάτων που συναντάμε στην βιβλιογραφία είναι το κβαντικό κόστος -quantum cost-, ο αριθμός των πυλών και ο αριθμός των γραμμών ελέγχου κυρίως στις πύλες Toffoli που είχαμε εξετάσει πριν. Για αυτό το λόγο, λοιπόν, έχουν προταθεί και αλγόριθμοι βελτιστοποίησης των αντίστροφων κυκλωμάτων όπως: α) Βελτιστοποίηση παραθύρου -Window optimization- που προτάθηκε από Soeken και Wille [14], και β) Μείωση αριθμού γραμμών ελέγχου όπως προτάθηκε από Wille, Soeken, Drechsler [16]. Τέλος, το δεύτερο μέρος προσφέρει επιπλέον αλγόριθμους επαλήθευσης, αποσφαλμάτωσης, δοκιμής και εξομοίωσης.

Τρίτο και τελευταίο μέρος της αρχιτεκτονικής του RevKit είναι οι εφαρμογές που μπορούν να υλοποιηθούν στη C++ και στην Python -και άρα μπορούν να είναι και command line applications. Οι εφαρμογές μπορούν να κάνουν χρήση όλων των πιο πάνω αλγορίθμων μέσω μιας διεπαφής (API).

Συνοψίζοντας, ένα παράδειγμα χρήσης του RevKit είναι η χρήση για λόγους μέτρησης επιδόσεων. Θα μπορούσε κανείς να φτιάξει μια εφαρμογή η οποία να δέχεται ως είσοδο μια μη αντίστροφη συνάρτηση η οποία περιγράφει ένα μη αντίστροφο κύκλωμα και, κάνοντας χρήση της ιεραρχικής μεθόδου από την μια και δυαδικού διαγράμματος από την άλλη, να επαληθεύσει τα αντίστροφα κυκλώματα που προκύπτουν και να τα συγκρίνει βάσει μετρικών που προαναφέρθηκαν. Για λόγους

σύγκρισης αποτελεσμάτων και ευκολότερης πρόσβασης στις αντίστροφες και μη συναρτήσεις, έχει δημιουργηθεί από τους Wille, Groβe, Teuber, W. Dueck και Drechsler μια ηλεκτρονική βάση που ονομάζεται RevLib [17]. Πέρα από βάση, το RevLib επίσης ορίζει το πρότυπο των αρχείων εισόδου και εξόδου που φιλοξενεί και τα οποία μπορούν επίσης να χρησιμοποιηθούν στο RevKit.



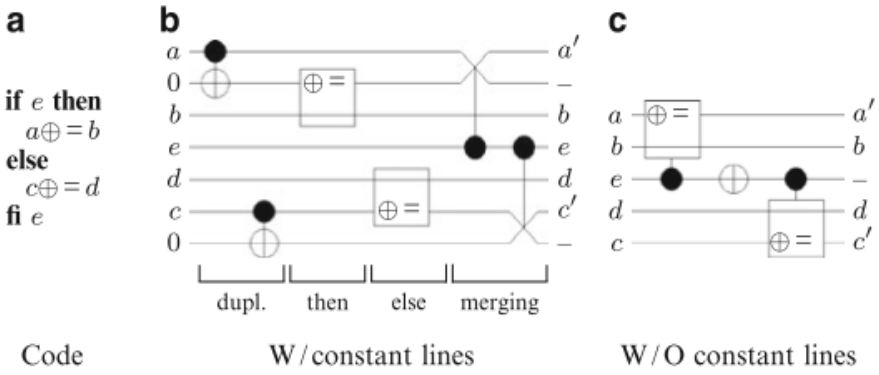
Σχήμα 2.6: Αρχιτεκτονική του RevKit [11]

2.3.6 SyReC: Synthesis of Reversible Circuits

Η γλώσσα προγραμματισμού SyReC [18] αναπτύχθηκε από τους Wille, Offermann και Drechsler και μπορεί να χρησιμοποιηθεί για να περιγράψει αντίστροφα συστήματα που πρόκειται να συντεθούν ως αντίστροφα κυκλώματα. Η σύνταξη των εντολών που χρησιμοποιεί βασίζεται πάνω στο σύνολο εντολών της αντίστροφης υψηλού επιπέδου γλώσσας προγραμματισμού Janus, η οποία θα εξεταστεί αργότερα και φαίνεται στο Σχήμα 2.7. Αρχικά, με τη βοήθεια των εντολών που δίνονται, περιγράφεται σε υψηλό επίπεδο η λειτουργία μιας μονάδας και στη συνέχεια, με μια ιεραρχική προσέγγιση το πρόγραμμα μετατρέπεται σε αντίστροφο κύκλωμα. Ένα παράδειγμα υλοποίησης μιας μονάδας ελέγχου ροής και μετατροπής της σε αντίστροφο κύκλωμα υπάρχει στο Σχήμα 2.8. Στην SyReC έχει υλοποιηθεί ένας απλός επεξεργαστής δεκαέξι bit ο οποίος θα συζητηθεί στα πλαίσια της αρχιτεκτονικής. Μετατροπές σε αντίστροφο κύκλωμα για όλες τις εντολές υπάρχουν στο [18].

(1) P ::= D* (**procedure** *id* S+)+
(2) D ::= *x* | *x* (*c*)
(3) V ::= *x* | *x.N:N* | *x.N*
(4) N ::= *c* | #V
(5) S ::= V <=> V | V ⊕= E |
(6) **if** E **then** S **else** S **fi** E |
(7) **from** *N* **do** S **loop** S **until** *N* |
(8) **for** *N* **do** S **until** *N* |
(9) **call** *id* | **uncall** *id* | **skip**
(10) E ::= *N* | V | (E ⊙ E) | (E ⊜ *N*)
(11) ⊕ ::= **+** | **-** | ^
(12) ⊙ ::= ⊕ ***** | **/** | **%** | ***/** | **&** | | | **&&** | || |
(13) < | > | = | **!=** | <= | >=
(14) ⊜ ::= << | >>

Σχήμα 2.7: Σύνταξη γλώσσας υλικού SyReC [18]



Σχήμα 2.8: Υλοποίηση μονάδας ελέγχου ροής[18]

2.4 Αντιστρεψιμότητα στην Αρχιτεκτονική Συνόλου Εντολών

Κατά τη διάρκεια της έρευνας του αντίστροφου υπολογισμού, μελετήθηκαν μερικές αρχιτεκτονικές που έχουν προταθεί, καθώς και μια γενική προσέγγιση στις απαιτούμενες λειτουργίες που θα πρέπει να υποστηρίζει. Πρωτοπόρες αρχιτεκτονικές θεωρούνται οι *"RISA" [19],* που βασίστηκε στην *"PDP-10"*, και η *"Pendulum instructions set architecture" [20]*. Σε αυτό το υποκεφάλαιο θα παρουσιαστούν δύο άλλες οι οποίες βασίζονται σε αυτές, η *"BobIsa reversible instruction set architecture"* [21] και ένας επεξεργαστής 16 bit με RISC αρχιτεκτονική που έχει υλοποιηθεί στη γλώσσα SyReC [22], η πρώτη είναι και η βάση της υλοποίησης του εξομοιωτή αργότερα στην διπλωματική εργασία. Πριν όμως από αυτό, είναι χρήσιμο να αναφερθούν μερικές επιλογές σχεδιασμού μιας αντίστροφης αρχιτεκτονικής που ισχύουν γενικά.

2.4.1 Επιλογές σχεδιασμού αντίστροφης αρχιτεκτονικής

Οι παραδοσιακές αρχιτεκτονικές που έχουν αναπτυχθεί τις τελευταίες δεκαετίες δεν υποστηρίζουν αντίστροφες εντολές, καθώς δεν υπήρξε τέτοια απαίτηση από το υλικό των επεξεργαστών. Μια αντίστροφη αρχιτεκτονική θα πρέπει, όμως, να παρέχει τη δυνατότητα συγγραφής αντίστροφων προγραμμάτων σε γλώσσες υψηλού επιπέδου που να στοχεύουν στην εκτέλεσή τους πάνω σε αντίστροφο υλικό και να εκμεταλλεύεται όλες τις δυνατότητές του. Τι χρειάζεται, λοιπόν, μια αντίστροφη αρχιτεκτονική;

Αρχικά, ας δούμε τις εντολές μνήμης. Στις γνωστές αρχιτεκτονικές οι τιμή μιας διεύθυνσης μνήμης μπορεί να καταστραφεί με μια αντικατάσταση από άλλη τιμή. Τέτοια προσέγγιση δεν μπορεί να ακολουθηθεί και στην αντίστροφη αρχιτεκτονική, επειδή είναι σημαντικό να διατηρήσουμε την πληροφορία για λόγους που έχουν εξηγηθεί νωρίτερα, αλλά και γιατί είναι πιθανό κάποια τιμή μνήμης που καταστράφηκε να χρειαστεί ανάκτηση με αντίστροφη εκτέλεση κάποια στιγμή αργότερα. Καταστροφικές πράξεις μνήμης δεν απαγορεύονται όμως πρέπει να αποφεύγονται από έναν προγραμματιστή και να δηλώνονται ρητά. Αντιθέτως, μια εντολή μνήμης που μπορεί να χρησιμοποιηθεί είναι να επιτρέπεται η εγγραφή στην μνήμη αν και μόνο αν ο

προορισμός περιέχει την τιμή μηδέν. Με τον τρόπο αυτό, μια αντίστροφη εκτέλεση είναι εύκολο να επιτευχθεί, αφού καμιά τιμή δεν καταστρέφεται. Δηλαδή, μια εντολή μνήμης θα πρέπει, για παράδειγμα, όταν διαβάζει μια τιμή από τη μνήμη να μηδενίζει τη διεύθυνση εκείνη για να αποφευχθούν καταστροφικές εντολές.

Κατά δεύτερο, θα εξεταστούν αριθμητικές εντολές αν και οι περισσότερες είναι αντίστροφες. Οι εντολές πρόσθεσης *"X=X+Y"* μπορούν εύκολα να αντιστραφούν κατά την αντίστροφη εκτέλεση με την εντολή *"X=X-Y"* και το ίδιο ισχύει για την αφαίρεση. Οι λογικοί τελεστές *"και"(and), "ή"(or) , "είτε"(xor)* και ο τελεστής άρνησης (*not)* μπορούν με τον ίδιο τρόπο να εκτελεστούν και αντίστροφα. Η εντολή ανάθεσης όπως την ξέρουμε είναι καταστροφική, και άρα δεν μπορεί να χρησιμοποιηθεί. Στη θέση της μπορεί να χρησιμοποιηθεί αντιμετάθεση (swap) με έναν καταχωρητή με μηδενική τιμή. Τα πράγματα δεν είναι τόσο εύκολα όταν έχουμε να κάνουμε με πράξεις κινητής υποδιαστολής *(floating point),* όπως ο πολλαπλασιασμός και η διαίρεση, για τον λόγο ότι οι πράξεις μετακίνησης, όπως shift left, shift right είναι καταστροφικές. Στη θέση τους σε συνδυασμό με άλλες τεχνικές μπορούν να χρησιμοποιηθούν πράξεις περιστροφής –left or right rotation.

Τέλος, είναι χρήσιμο να αναφερθούμε σε εντολές μετακίνησης jump. Η πληροφορία που χρειάζεται στους κλασικούς επεξεργαστές για την εκτέλεση της εντολής jump δεν είναι αρκετή για την αντίστροφη αρχιτεκτονική, επειδή κατά την εκτέλεση της εντολής δεν είναι γνωστό ποια κατεύθυνση ακολουθείται, προς τα εμπρος ή προς τα πίσω, και ούτε είναι γνωστό αν το jump εκτελείται για να μεταφερθεί στο σημείο από το οποίο έφτασε εκεί. Μια λύση για αυτό είναι η εκτέλεση τέτοιων εντολών να γίνεται βάσει άλλων δύο καταχωρητών: του καταχωρητή κατεύθυνσης και του καταχωρητή διακλάδωσης *(branch).* Σε συνδυασμό με τους τρεις καταχωρητές η πληροφορία είναι αρκετή ώστε η εντολή jump να παρέχει και την αντιστρεψιμότητα.

2.4.2 Επεξεργαστής αρχιτεκτονικής RISC υλοποιημένος με αντίστροφη λογική

Λόγω της ανάπτυξης της γλώσσας προγραμματισμού SyReC, έγινε πιο εύκολη η προσπάθεια προσομοίωσης αντίστροφων κυκλωμάτων, και συνεπώς αναπτύχθηκε με την συγκεκριμένη γλώσσα ένας αντίστροφος επεξεργαστής [22]. Ο επεξεργαστής

κατασκευάστηκε σύμφωνα με την Harvard αρχιτεκτονική και δεν υποστηρίζει αντίστροφες εντολές, όμως μπορεί να εκτελέσει προγράμματα με τις εντολές που φαίνονται στο Σχήμα 2.9 σε ένα αντίστροφο υλικό πάνω σε εξομοιωτή. Ο επεξεργαστής υπάρχει στον ιστότοπο RevLib [17] και μπορεί να προσομοιωθεί στον RevKit [11], αλλά όχι εξ ολοκλήρου, καθώς η μνήμη προγράμματος και η μνήμη δεδομένων έχουν υλοποιηθεί σαν εξωτερικό κομμάτι στην γλώσσα Python και είναι ένα ανοιχτό ζήτημα η υλοποίησή τους στην SyReC. Μια αρχική σκέψη για τον σκοπό της διπλωματικής εργασίας ήταν η υλοποίηση τους ή η εξομοίωση προγραμμάτων για ερευνητικούς σκοπούς, όμως τελικά αυτή απορρίφθηκε και για το λόγο αυτό δεν είναι απαραίτητη η περαιτέρω παρουσίαση του συγκεκριμένου έργου.

Command	Semantic
Arithmetic and Logic Instructions	
ADC $R[i]$, $R[j]$, $R[k]$	Addition with carry into $R[i]$
SBC $R[i]$, $R[j]$, $R[k]$	Substraction with carry into $R[i]$
ADD $R[i]$, $R[j]$, $R[k]$	Addition without carry into $R[i]$
SUB $R[i]$, $R[j]$, $R[k]$	Substraction without carry into $R[i]$
ROR $R[i]$, $R[j]$	Bitrotation right of $R[j]$
ROL $R[i]$, $R[j]$	Bitrotation left of $R[j]$
SHR $R[i]$, $R[j]$	Bitshift right of $R[j]$
SHL $R[i]$, $R[j]$	Bitshift left of $R[j]$
NOT $R[i]$, $R[j]$	Bitwise negation
XOR $R[i]$, $R[j]$, $R[k]$	Bitwise exor
OR $R[i]$, $R[j]$, $R[k]$	Bitwise or
AND $R[i]$, $R[j]$, $R[k]$	Bitwise and
MKB $R[i]$, $R[j]$, b	Masking of bit b
INB $R[i]$, $R[j]$, b	Inverting of bit b
SEB $R[i]$, $R[j]$, b	Set bit b
CLB $R[i]$, $R[j]$, b	Clear bit b
Jump Instructions	
JMP d	Jump to address d
JC d	Jump to address d, if carry is set
JZ d	Jump to address d, if zero-flag is set
JNC d	Jump to address d, if carry is not set
JNZ d	Jump to address d, if zero-flag is not set
Load/Store Instructions	
LDD $R[i]$, $R[j]$	Load memory content of address $R[j]$ into $R[i]$
STO $R[j]$, $R[k]$	Store $R[k]$ into memory at address $R[j]$
LDL $R[i]$, d	Load constant d into low-byte of $R[i]$
LDH $R[i]$, d	Load constant d into high-byte of $R[i]$

Σχήμα 2.9: Εντολές αντίστροφου επεξεργαστή αρχιτεκτονικής Harvard. [22]

2.5 Αντίστροφες γλώσσες προγραμματισμού

Στα πλαίσια της διπλωματικής εργασίας και για σκοπούς καλύτερης κατανόησης του αντίστροφου υπολογισμού, μελετήθηκαν επίσης θέματα που έχουν να κάνουν με υψηλού επιπέδου αντίστροφες γλώσσες. Μερικές αντίστροφες γλώσσες προγραμματισμού είναι οι R, ESRL, SRL, EPA και τελικά η Janus. Στην τελευταία έγινε η περισσότερη μελέτη, καθώς είναι αυτή που χρησιμοποιείται περισσότερο στις αναφορές αυτού του πεδίου. Η γλώσσα Janus προτάθηκε από τους Lutz και Derby στο γράμμα [23] που απέστειλαν στον Landauer το 1986. Όπως και κάθε αντίστροφη γλώσσα θα πρέπει να διατηρεί συγκεκριμένες δομές που επιτρέπουν την αντίστροφη εκτέλεση, τέτοιες δομές της Janus υπάρχουν στο Σχήμα 2.10. Για παράδειγμα, για την εκτέλεση εντολών διακλάδωσης και εντολών επανάληψης χρειάζονται μετασυνθήκες όπως είναι γνωστές από τον συναρτησιακό προγραμματισμό (functional or logical programming), έτσι ώστε η μετασυνθήκη κατά την αντίστροφη εκτέλεση να ορίσει την ροή της διακλάδωσης ή να σταματήσει την επανάληψη αντίστοιχα. Για να καλούμε συναρτήσεις για αντίστροφη εκτέλεση χρειάζεται η νέα εντολή *“UNCALL”* που είναι αντίστροφη της *“CALL”*. Οι εντολές εισόδου εξόδου *“READ-WRITE”* ορίζονται επίσης ως η μία αντίστροφη της άλλης. Άλλες πράξεις, όπως οι αριθμητικές, είναι αντίστροφες με τον ίδιο τρόπο που εξετάστηκαν στα πλαίσια της αρχιτεκτονικής. Περισσότερες λεπτομέρειες δεν είναι απαραίτητες για τους σκοπούς της διπλωματικής εργασίας και δεν χρειάζεται να δοθούν.

Instruction	**Inverse**
$stmt_1\ stmt_2$	$[stmt_2]^{-1}\ [stmt_1]^{-1}$
$\overset{pre}{\texttt{IF}}\ condition\ \texttt{THEN}\ \overset{if}{stmt}\ \texttt{ELSE}\ \overset{else}{stmt}\ \overset{post}{\texttt{ENDIF}}\ condition$	$\overset{post}{\texttt{IF}}\ condition\ \texttt{THEN}\ \left[\overset{if}{stmt}\right]^{-1}\ \texttt{ELSE}\ \left[\overset{else}{stmt}\right]^{-1}_{pre}\ \texttt{ENDIF}\ condition$
$\overset{from}{\texttt{FROM}}\ condition\ \overset{do}{\texttt{DO}}\ stmt\ \overset{loop}{\texttt{LOOP}}\ stmt\ \overset{until}{\texttt{UNTIL}}\ condition$	$\overset{until}{\texttt{FROM}}\ condition\ \texttt{DO}\ \left[\overset{do}{stmt}\right]^{-1}\ \texttt{LOOP}\ \left[\overset{loop}{stmt}\right]^{-1}_{from}\ \texttt{UNTIL}\ condition$
$\texttt{CALL}\ name$	$\texttt{UNCALL}\ name$
$\texttt{UNCALL}\ name$	$\texttt{CALL}\ name$
$\overset{1}{var}\ \texttt{:}\ \overset{2}{var}$	$\overset{1}{var}\ \texttt{:}\ \overset{2}{var}$
$name\ \texttt{+=}\ expression$	$name\ \texttt{-=}\ expression$
$name\ \texttt{-=}\ expression$	$name\ \texttt{+=}\ expression$
$name\ \texttt{\^{}=}\ expression$	$name\ \texttt{\^{}=}\ expression$
$\texttt{READ}\ name$	$\texttt{WRITE}\ name$
$\texttt{WRITE}\ name$	$\texttt{READ}\ name$

Σχήμα 2.10: Αντιστροφές για εντολές γλώσσας Janus. [2]

Κεφάλαιο 3

Αντίστροφος Προγραμματισμός

3.1 BobIsa Reversible Instruction Set Architecture

Η συγκεκριμένη αρχιτεκτονική βασίζεται πάνω στην “Pendulum Instruction Set Architecture” και παρουσιάζεται από τον Michael Kirkedal στο [21]. Πάνω στην αρχιτεκτονική αυτή βασίζεται και το σύνολο εντολών του προσομοιωτή που αναπτύσσεται στα πλαίσια της διπλωματικής εργασίας. Περαιτέρω λεπτομέρειες προγραμματισμού μπορείτε να βρείτε στο [21]. Σε αυτό το κεφάλαιο, θα αναφερθούν οι εντολές που υποστηρίζει οι οποίες τηρούν όλες τις προδιαγραφές που έχουμε θέσει στο υποκεφάλαιο 2.4.1.

3.2 Εντολές Μνήμης

Η αρχιτεκτονική υποστηρίζει μόνο μία εντολή μνήμης που φαίνεται στο Σχήμα 3.1., όπου το $reg_s d$ περιέχει την τιμή που πρόκειται να αποθηκεύσουμε και το reg_{adr} τη διεύθυνση μνήμης στην οποία πρόκειται η τιμή να γραφτεί. Η εντολή είναι μη καταστροφική, καθώς η τιμή που βρίσκεται στην διεύθυνση reg_{adr} ανταλλάσσεται με την τιμή που περιέχει το $reg_s d$.

i			$inv(i)$		
EXCH	$reg_s d$	reg_{adr}	EXCH	$reg_s d$	reg_{adr}

Σχήμα 3.1: Εντολή μνήμης και η αντίστροφή της. [21]

3.3 Αριθμητικές Εντολές

Οι αριθμητικές εντολές φαίνονται στο Σχήμα 3.2 και οι αντίστροφές τους έχουν τη λογική που εξηγήθηκε στο προηγούμενο υποκεφάλαιο. Όλες οι εντολές είναι αντιστρέψιμες χωρίς καμιά ιδιαίτερη υλοποίηση, εκτός από τον πολλαπλασιασμό και τη διαίρεση, που υλοποιήθηκαν ώστε να ισχύει ότι η μια εντολή είναι αντίστροφη της άλλης. Η εντολή της άρνησης είναι αντίστροφη του εαυτού της.

i			$inv(i)$		
ADD	reg_d	reg_s	SUB	reg_d	reg_s
SUB	reg_d	reg_s	ADD	reg_d	reg_s
ADD1	reg_d		SUB1	reg_d	
SUB1	reg_d		ADD1	reg_d	
NEG	reg_d		NEG	reg_d	
XOR	reg_d	reg_s	XOR	reg_d	reg_s
XORI	reg_d	imm	XORI	reg_d	imm
MUL2	reg_d		DIV2	reg_d	
DIV2	reg_d		MUL2	reg_d	

Σχήμα 3.2: Αριθμητικές εντολές και οι αντίστροφές τους. [21]

3.4 Εντολές Ελέγχου Ροής

Οι εντολές ελέγχου ροής φαίνονται στο Σχήμα 3.3 και χρησιμοποιούνται απαράλλακτες και ως αντίστροφες. Οι εντολές αυτές υπολογίζονται βάσει των τριών καταχωρητών που αναφέραμε νωρίτερα και η μόνη εντολή που αλλάζει την κατεύθυνση της ροής είναι η *"RBRA"*. Για όλες τις εντολές ισχύει ότι μετακινούνται κατά *"off"* όταν η εκτέλεση γίνεται προς τα προς και κατά *"-off"* όταν γίνεται προς τα πίσω. Η εντολή SWBR είναι εντολή ανταλλαγής του reg_{ds} με τον καταχωρητή διακλάδωσης.

i			$inv(i)$		
BGEZ	reg_d	off	BGEZ	reg_d	off
BLZ	reg_d	off	BLZ	reg_d	off
BEVN	reg_d	off	BEVN	reg_d	off
BODD	reg_d	off	BODD	reg_d	off
BRA		off	BRA		off
RBRA		off	RBRA		off
SWBR	reg_{ds}		SWBR	reg_{ds}	

Σχήμα 3.3: Εντολές διακλάδωσης και οι αντίστροφές τους. [21]

3.5 Στοιχεία αρχιτεκτονικής

Όσον αφορά την μορφή των εντολών, το μέγεθος της μνήμης και άλλα στοιχεία υλοποίησης αυτά φαίνονται στο Σχήμα 3.4 και στον Πίνακα 3.1 αντίστοιχα. Η κωδικοποίηση των αριθμών γίνεται με 2's compliment και ο καταχωρητής μηδέν πάντα έχει την τιμή μηδέν.

bits:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	1	1	0	0	reg_d				reg_s				0	1	0	0
SUB	1	1	0	0	reg_d				reg_s				1	1	0	1
ADD1	1	1	0	0	reg_d				0	0	0	0	0	1	1	0
SUB1	1	1	0	0	reg_d				0	0	0	0	1	1	1	1
NEG	1	1	0	0	reg_d				0	0	0	0	0	1	1	1
XOR	1	1	0	0	reg_d				reg_s				0	0	0	0
XORI	0	0	0	0	reg_d				imm							
MUL2	1	0	1	0	reg_d				0	0	0	0	0	0	0	0
DIV2	1	0	0	1	reg_d				0	0	0	0	0	0	0	0
EXCH	1	0	0	0	reg_d				reg_{adr}				0	0	0	0
BGEZ	0	0	1	1	reg_d				off							
BLZ	0	0	1	0	reg_d				off							
BEVN	0	1	0	1	reg_d				off							
BODD	0	1	0	0	reg_d				off							
BRA	0	0	0	1	0	0	0	0	off							
RBRA	0	1	1	1	0	0	0	0	off							
SWBR	0	1	1	0	reg_d				0	0	0	0	0	0	0	0

Σχήμα 3.4: Κωδικοποίηση των εντολών. [21]

	Μέγιστη τιμή αποθήκευσης σε bit	Ποσότητα	Εύρος τιμών
Καταχωρητές	16 σε ένα καταχωρ.	16 καταχωρητές	-32768 μέχρι 32767
Μνήμη	16 σε μια λέξη	2^{16} λέξεις	-32768 μέχρι 32767
Offset μετακίνησης (jumps)	8	127 γραμμές το πολύ	-
Σταθεροί αριθμοί (immediates)	8	-	-128 μέχρι 127

Πίνακας 3.1: Στοιχεία υλοποίησης.

Κεφάλαιο 4

Το Εργαλείο

4.1 Οργάνωση

Το κεφάλαιο αυτό έχει σκοπό την παρουσίαση του γραφικού περιβάλλοντος του εξομοιωτή, καθώς και την παρουσίαση κυρίων των λειτουργιών του. Στα παρακάτω υποκεφάλαια θα γίνει λεπτομερής ανασκόπηση όλων των επιλογών που παρέχονται από το γραφικό περιβάλλον διεπαφής με τον χρήστη, καθώς και λειτουργιών όπως είναι η εκτέλεση, η αποσφαλμάτωση προγραμμάτων αλλά και επιπλέον δυνατοτήτων.

4.2 Συμβατότητα

Ο εξομοιωτής υλοποιήθηκε στην υψηλού επιπέδου γλώσσα προγραμματισμού C και το γραφικό περιβάλλον υλοποιήθηκε με την βοήθεια της εργαλειοθήκης GTK, η οποία παρέχει διεπαφή -API- στην γλώσσα C. Ο σχεδιασμός του γραφικού περιβάλλοντος έγινε με την εφαρμογή "Glade–A User Interface Designer" [26]. Η

ανάπτυξη του εξομοιωτή έγινε στον λειτουργικό σύστημα Linux Ubuntu με γραφικό περιβάλλον GNOME. Οι εκδόσεις των πιο πάνω εργαλείων βρίσκονται στον Πίνακα 6.1.

Εργαλείο	**Έκδοση**
Μεταγλωττιστής gcc	5.2.1
GTK	3.0
Glade – A User Interface Designer	3.18.3
Linux Ubuntu	15.10
GNOME	3.0

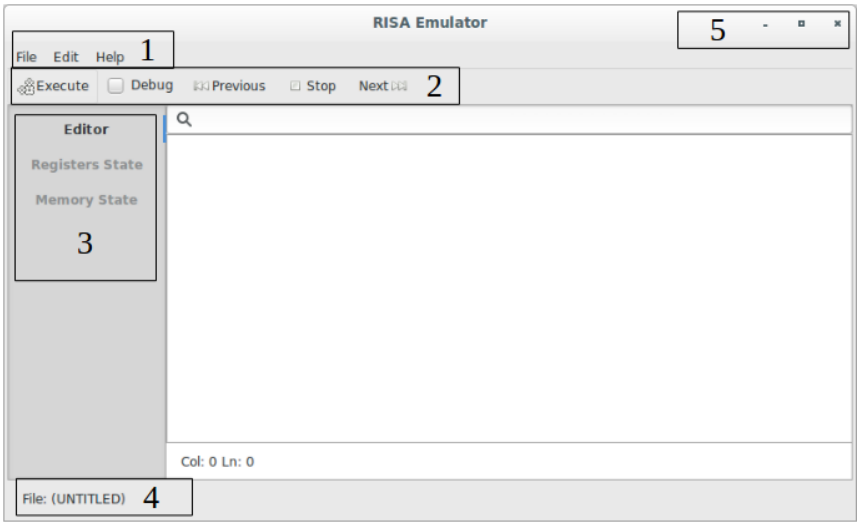
Πίνακας 6.1: Εκδόσεις Εργαλείων.

4.3 Γραφικό Περιβάλλον

Σε αυτό το υποκεφάλαιο θα παρουσιάσω το γραφικό περιβάλλον διεπαφής του χρήστη με την εφαρμογή. Αρχικά, θα δείξω την εμφάνιση του εξομοιωτή δηλαδή τις επιλογές που είναι προσβάσιμες από οποιοδήποτε εσωτερικό παράθυρο του γραφικού περιβάλλοντος, έπειτα θα παρουσιάσω τα εσωτερικά παράθυρα του εξομοιωτή όπως ο κειμενογράφος, η προβολή των καταχωρητών και η προβολή της μνήμης.

4.3.1 Εμφάνιση

Στο Σχήμα 4.1 μπορούμε να δούμε τις επιλογές που προσφέρονται στο εργαλείο. Στο πλαίσιο 1 βλέπουμε το μενού που έχει τρεις επιλογές. Η επιλογή "*File*" περιέχει λειτουργίες οι οποίες σχετίζονται με την διαχείριση των αρχείων, δηλαδή των προγραμμάτων που επεξεργαζόμαστε στον κειμενογράφο. Η επιλογή *"Edit"* περιέχει λειτουργίες που σχετίζονται με την επεξεργασία κειμένου. Με την επιλογή *"Help"* μπορούμε να δούμε γενικές πληροφορίες που αφορούν την εφαρμογή.



Σχήμα 4.1: Εμφάνιση

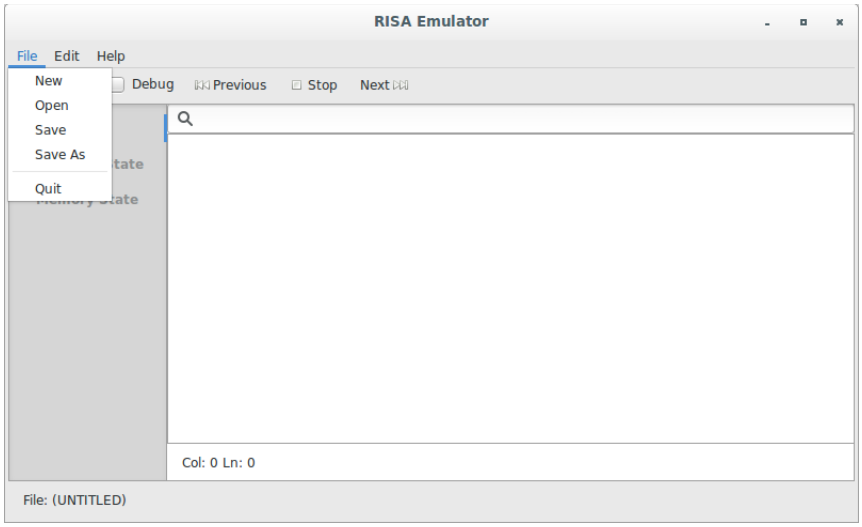
Στο πλαίσιο 2 φαίνονται οι εντολές που μπορούμε να δώσουμε στον εξομοιωτή. Η εντολή *"Execute"* είναι εντολή εκτέλεσης προγράμματος που βρίσκεται στον κειμενογράφο με αποσφαλμάτωση ή χωρίς. Η εντολή *"Debug"* είναι εντολή αποσφαλμάτωσης και χρησιμοποιείται σε συνδυασμό με την εντολή "Execute" του προγράμματος που βρίσκεται στον κειμενογράφο. Η εντολή *"Previous"* χρησιμοποιείται κατά την αποσφαλμάτωση και εκτελεί την προηγούμενη εντολή του προγράμματος. Η εντολή *"Next"* χρησιμοποιείται κατά την αποσφαλμάτωση και εκτελεί την επόμενη εντολή του προγράμματος. Η εντολή "Stop" σταμάτα την διαδικασία αποσφαλμάτωση.

Στο πλαίσιο 3 βρίσκεται η εναλλαγή προβολών. Υπάρχουν τρεις διαθέσιμες προβολές: το *"Editor"* (προβολή κειμενογράφου), το *"Register State" (*προβολή κατάστασης καταχωρητών) και τέλος, το *"Memory state" (*προβολή κατάστασης μνήμης).

Τέλος στο πλαίσιο 4 φαίνεται το όνομα του αρχείου στο οποίο δουλεύουμε και στο πλαίσιο 5 η απόκρυψη, η μεγιστοποίηση και το κλείσιμο παραθύρου.

4.3.2 Μενού Επιλογών

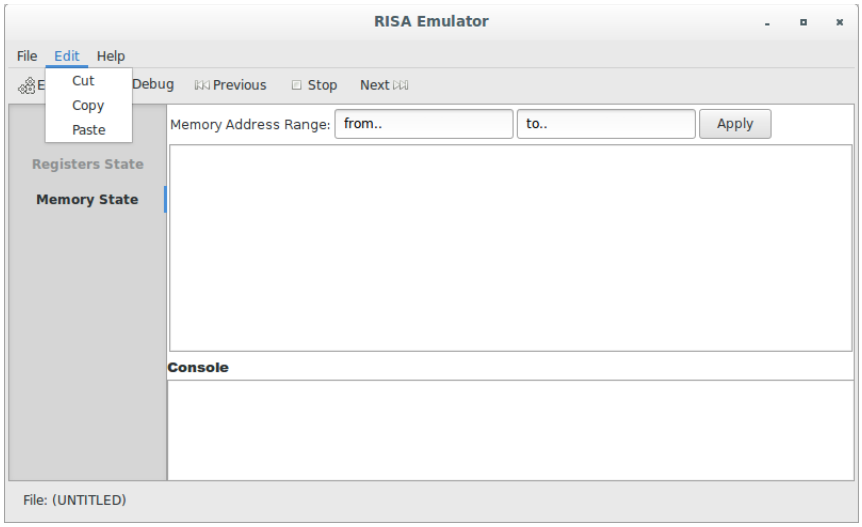
Στο Σχήμα 4.2 μπορούμε να δούμε τις επιλογές που σχετίζονται με τη διαχείριση αρχείων. Η επιλογή *"New"* δημιουργεί νέο αρχείο, η επιλογή *"Open"* ανοίγει ένα παράθυρο εξερεύνησης αρχείων για να επιλέξουμε ένα αρχείο προς άνοιγμα, η επιλογή *"Save"* και η επιλογή *"Save As"* ανοίγουν παράθυρο εξερεύνησης αρχείων ώστε να επιλέξουμε τον κατάλογο στον οποίο θέλουμε να αποθηκεύσουμε το αρχείο μας. Τέλος, η επιλογή *"Quit"* κλείνει την εφαρμογή.



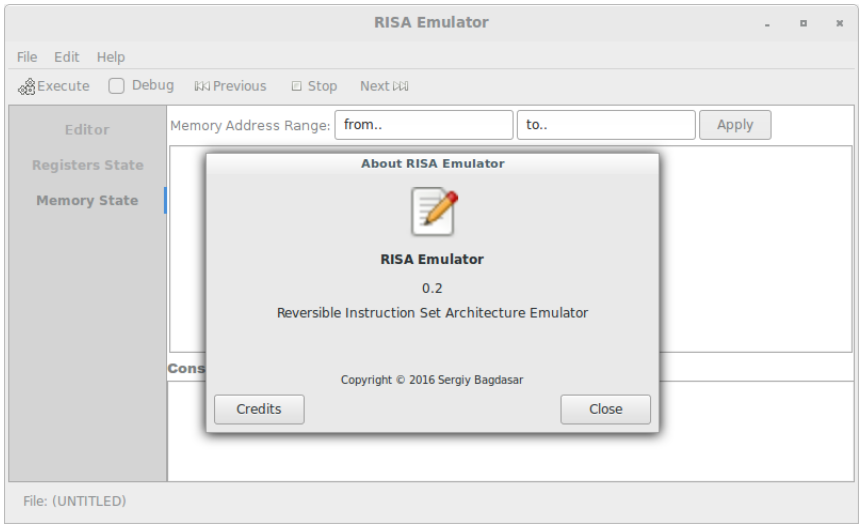
Σχήμα 4.2: Μενού Επιλογών Αρχείου

Στο Σχήμα 4.3 μπορούμε να δούμε τις επιλογές που σχετίζονται με τη διαχείριση κειμένου. Συγκεκριμένα, οι επιλογές *"Cut"*, *"Copy"* και *"Paste"* αποκόπτουν, αντιγράφουν και επικολλούν ένα επιλεγμένο μέρος κειμένου στον κειμενογράφο αντίστοιχα.

Τέλος, στο Σχήμα 4.4 βλέπουμε χρήσιμες πληροφορίες, όπως είναι η έκδοση της εφαρμογής, το όνομα του προγραμματιστή καθώς και η ηλεκτρονική διεύθυνση επικοινωνίας (πατώντας πάνω στο όνομα, στην επιλογή *"Credit")*.



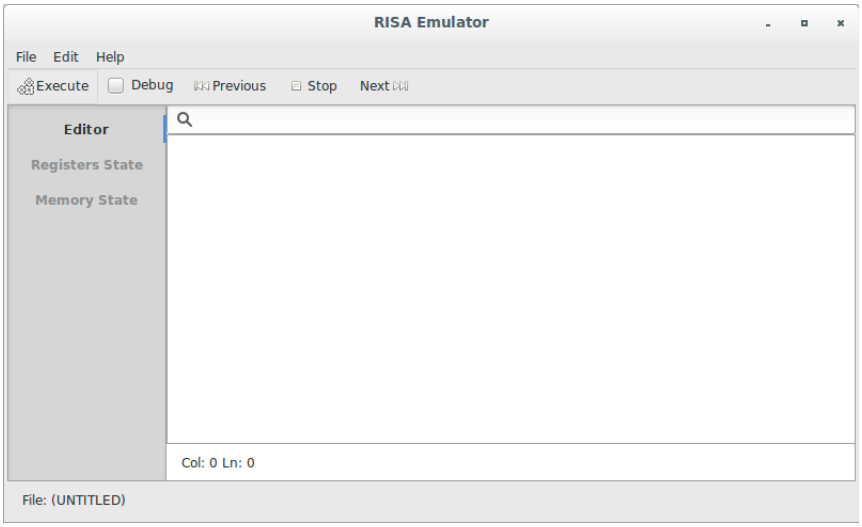
Σχήμα 4.3: Μενού Επιλογών Κειμένου



Σχήμα 4.4: Μενού Επιλογών Βοήθειας

4.3.3 Κειμενογράφος

Στο Σχήμα 4.5 μπορούμε να δούμε τον κειμενογράφο. Στον κειμενογράφο μπορούμε να αναπτύξουμε τα προγράμματα μας και στην συνέχεια να τα εκτελέσουμε και, αν χρειαστεί, να τα αποσφαλματώσουμε. Ο κειμενογράφος προσφέρει βασικές δυνατότητες όπως η αναζήτηση -πάνω μέρος- μέσα στο πρόγραμμα και η αναγραφή της στήλης και της γραμμής που βρισκόμαστε ανά πάσα στιγμή.



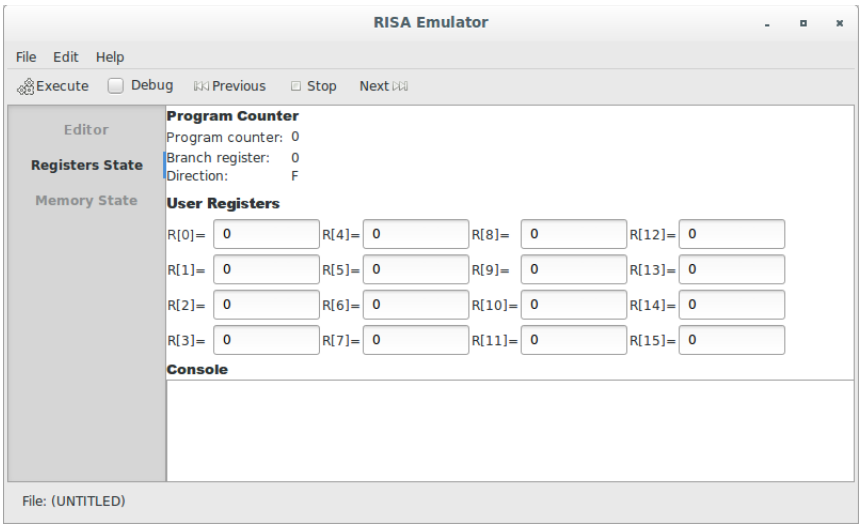
Σχήμα 4.5: Κειμενογράφος

4.3.4 Καταχωρητές και κονσόλα

Στο Σχήμα 4.6 φαίνεται η προβολή των καταχωρητών. Σε αυτή την προβολή έχουμε την δυνατότητα να παρακολουθήσουμε τις καταστάσεις όλων των καταχωρητών κατά την διάρκεια της αποσφαλμάτωσης ή στο τέλος μιας εκτέλεσης. Στο πάνω μέρος της προβολής -τμήμα *"Program Counter"*- βλέπουμε τον μετρητή προγράμματος -program counter- τον καταχωρητή βρόγχων -branch register-, καθώς και την κατεύθυνση στην οποία εκτελείται το πρόγραμμα μετά από κάθε εντολή που εκτελείται. Οι τρεις αυτοί καταχωρητές υπολογίζονται σύμφωνα με τον ορισμό που δίνεται από τον M.Kirkedal στο [21].

Οι υπόλοιποι καταχωρητές κοινής χρήσης φαίνονται στο τμήμα *"User Registers"*, όπου μπορούμε να παρακολουθήσουμε τις καταστάσεις όλων των καταχωρητών που μπορεί να αξιοποιήσει ο προγραμματιστής κατά την ανάπτυξη ενός προγράμματος.

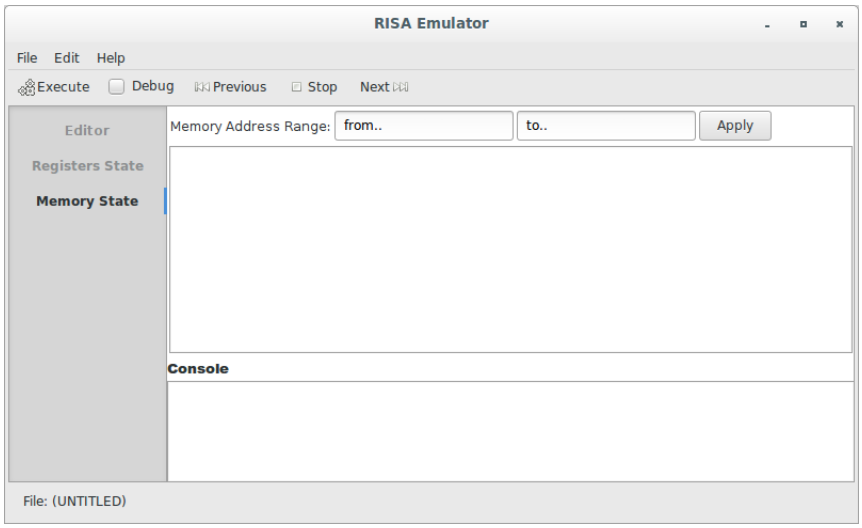
Τέλος, βλέπουμε το τελευταίο τμήμα της συγκεκριμένης προβολής -που είναι το ίδιο στην προβολή της μνήμης-, την κονσόλα -console-. Η κονσόλα είναι πολύ χρήσιμη, αφού εμφανίζει πληροφορίες σχετικά με συντακτικά λάθη ή λάθη κατά την εκτέλεση, ενώ επιπλέον δείχνει την εντολή που εκτελείται κατά την αποσφαλμάτωση και τις δύο εντολές που είναι γύρω της, εφόσον αυτές υπάρχουν.



Σχήμα 4.6: Καταχωρητές

4.3.5 Μνήμη

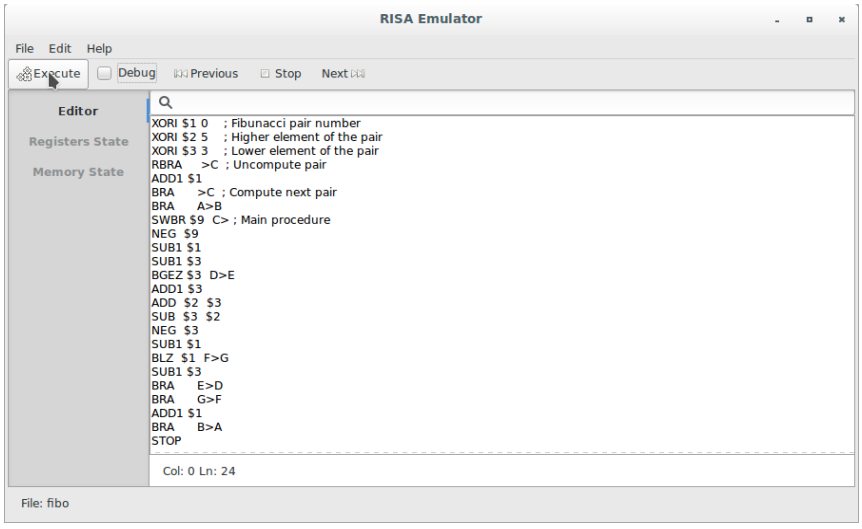
Σε αυτή την προβολή μπορούμε να δούμε τις τιμές που έχει η μνήμη του εξομοιωτή σε συγκεκριμένες διευθύνσεις. βάζοντας το εύρος των διευθύνσεων στα κουτιά *"from"* -*"από"*- και *"to"* -*"μέχρι"*- και πατώντας το κουμπί *"Apply"*. Οι τιμές των διευθύνσεων δεν ανανεώνονται αυτόματα, αλλά θα πρέπει να ξαναπατήσετε το κουμπί *"Apply"* μετά από κάποια αλλαγή κατάστασης. Η κονσόλα σε αυτή την προβολή δείχνει τις ίδιες πληροφορίες με την κονσόλα στην προβολή των καταχωρητών.



Σχήμα 4.7: Μνήμη

4.4 Λειτουργίες, Ανάπτυξη Προγραμμάτων

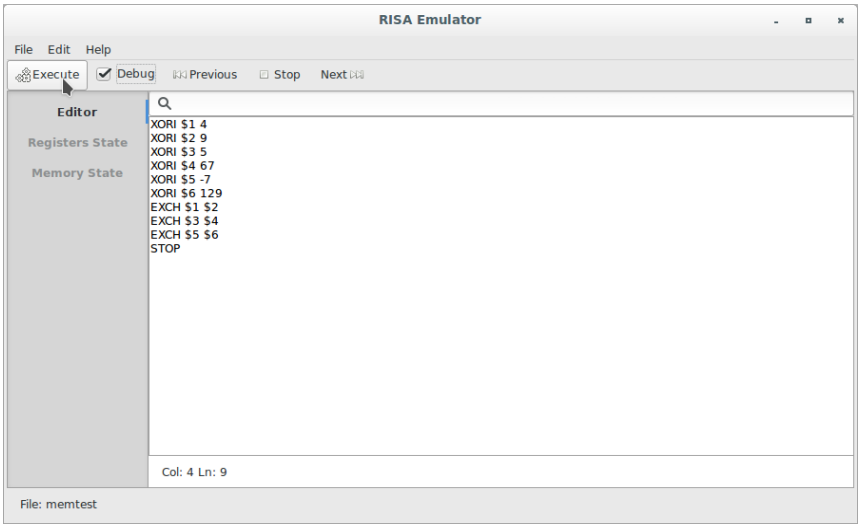
Η πρώτη κύρια λειτουργία του εξομοιωτή είναι η δυνατότητα να γράψουμε και να εκτελέσουμε προγράμματα. Όπως βλέπουμε στο Σχήμα 4.8, μπορούμε γράφοντας το πρόγραμμά μας στον κειμενογράφο να το εκτελέσουμε.



Σχήμα 4.8: Εκτέλεση Προγράμματος

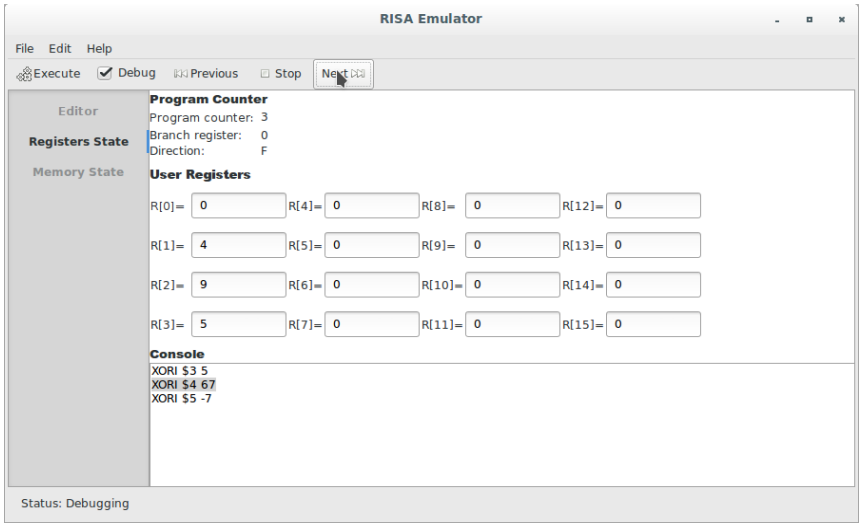
Τα αποτελέσματα εκτέλεσης των προγραμμάτων ανάλογα με την φύση του καθενός και της χρήσης των καταχωρητών και της μνήμης μπορούν να βρεθούν με την προβολή των καταχωρητών ή την προβολή της μνήμης με τον τρόπο που εξηγήθηκε σε προηγούμενα υποκεφάλαια.

Η δεύτερη κύρια λειτουργία του εξομοιωτή είναι η αποσφαλμάτωση των προγραμμάτων. Από το Σχήμα 4.10 φαίνεται ο τρόπος που μπορούμε να ξεκινήσουμε μια αποσφαλμάτωση.

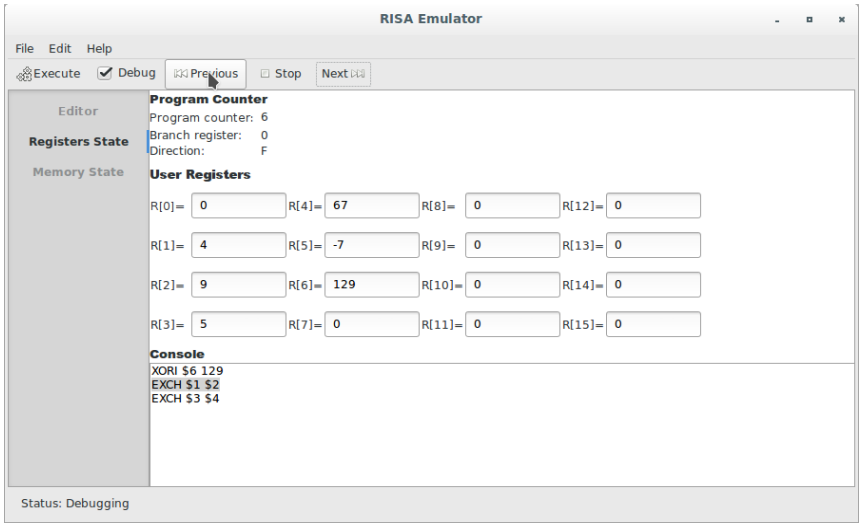


Σχήμα 4.10: Αποσφαλμάτωση Προγράμματος

Ο εξομοιωτής, λόγω της φύσης του, δεν προσφέρει μόνο την συνηθισμένη αποσφαλμάτωση προς τα μπροστά (Σχήμα 4.11), αλλά και προς τα πίσω (Σχήμα 4.12), χωρίς καμία επιπλέον δομή αποθήκευσης της ροής εκτέλεσης, εκμεταλλευόμενος τη δυνατότητα εκτέλεσης και προς τις δύο πλευρές που προσφέρει ο αντίστροφος υπολογισμός.



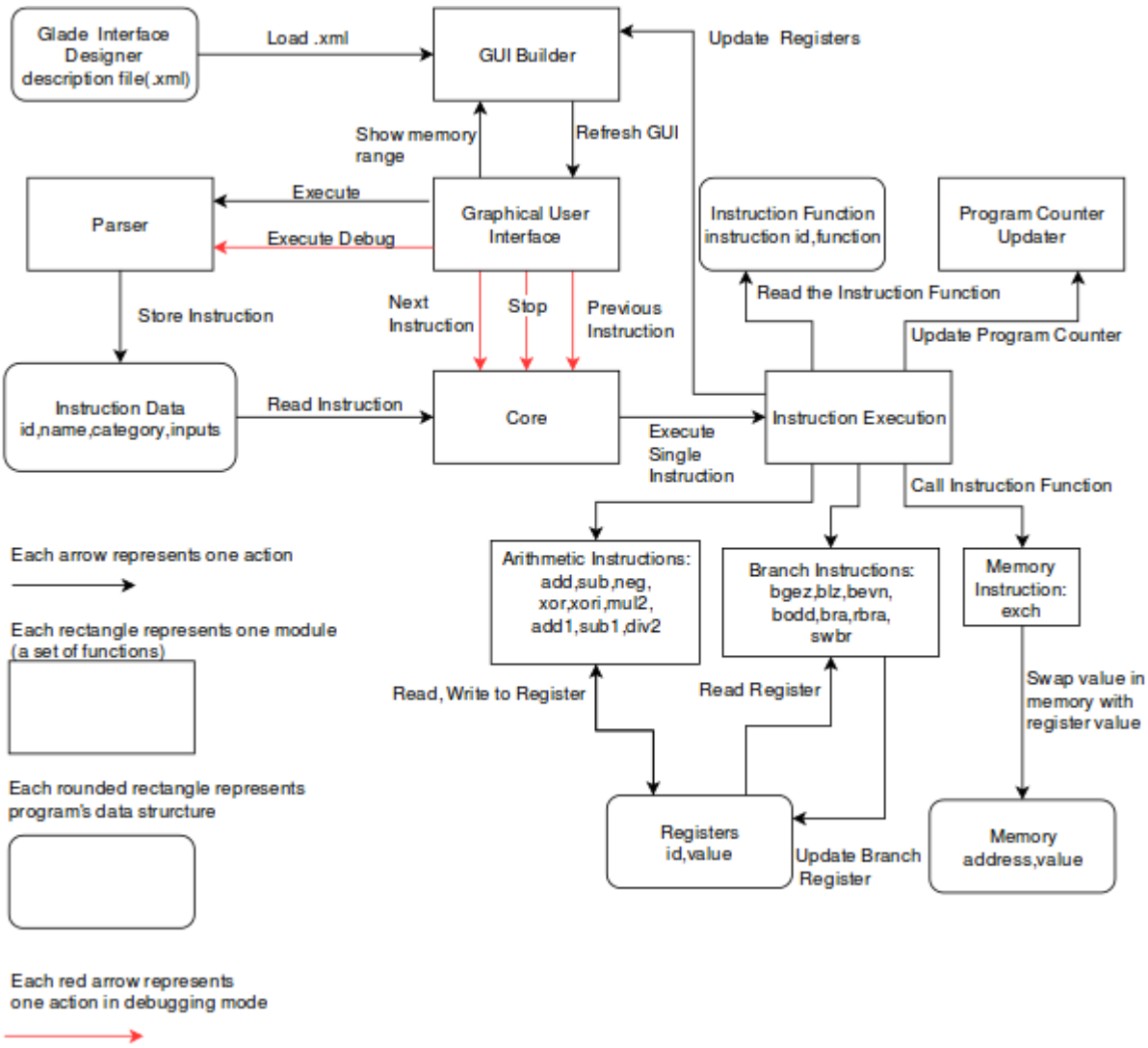
Σχήμα 4.11: Αποσφαλμάτωση Μπροστά



Σχήμα 4.12: Αποσφαλμάτωση Πίσω

Επιπλέον λειτουργία του προσομοιωτή είναι και η αλλαγή της τιμής ενός καταχωρητή κατά τη διάρκεια μιας αποσφαλμάτωσης. Κάτι τέτοιο βοηθάει να διορθώσουμε τυχόν λάθη και να μην σταματάμε τη διαδικασία αποσφαλμάτωσης κάθε φορά. Μπορούμε να αλλάξουμε την τιμή ενός καταχωρητή πατώντας πάνω του στην προβολή των καταχωρητών. Μια ακόμα λειτουργία του εξομοιωτή είναι η αναγνώριση συντακτικών λαθών στο πρόγραμμα για ευνόητους λόγους. Παραδείγματα χρήσης των παραπάνω δυνατοτήτων δίνονται στο Κεφάλαιο 5.

4.5 Υλοποίηση “RISA Emulator”



Σχήμα 4.13: Διάγραμμα Εξομοιωτή

Σε αυτό το υποκεφάλαιο θα γίνει αναφορά στον τρόπο υλοποίησης του εξομοιωτή. Στο Σχήμα 4.13 μπορούμε να δούμε τη ροή του προγράμματος και τα δεδομένα που χρησιμοποιεί και αποθηκεύει κατά την εκτέλεση ή την αποσφαλμάτωση ενός αντίστροφου προγράμματος. Όπως φαίνεται, οι εντολές λαμβάνονται από το γραφικό περιβάλλον του εξομοιωτή το οποίο έχει φορτωμένο ένα αντίστροφο πρόγραμμα μέσα στον κειμενογράφο. Πιο κάτω θα εξηγήσω τον τρόπο που

υλοποιήθηκε η εκτέλεση των εντολών, καθώς επίσης και τον τρόπο υλοποίησης των μονάδων (modules) βάσει του πάνω σχήματος.

Όπως παρουσιάσαμε ήδη, υπάρχουν δύο λειτουργίες που ξεκινούν τον εξομοιωτή: η εκτέλεση και η αποσφαλμάτωση. Όταν δοθούν οι εντολές για τις παραπάνω λειτουργίες, το αντίστροφο πρόγραμμα φορτώνεται σε μια ενδιάμεση μνήμη (buffer) και αρχίζει η διαδικασία της ανάλυσης (parsing). Στη συνέχεια, το κείμενο μέσα στον buffer χωρίζεται ανά γραμμή, όπως φαίνεται στο Σχήμα 4.14 όπου κάθε γραμμή είναι μια εντολή. Έπειτα, η κάθε εντολή αναλύεται ώστε να καταγράψουμε την κατηγορία στην οποία ανήκει ανάλογα με τις εισόδους που παίρνει και τον μοναδικό αριθμό της (id). Τελικά, αναλύουμε τις εισόδους της εντολής που μπορούν να είναι καταχωρητές, σταθεροί αριθμοί, γραμμές μετακίνησης ή μια ετικέτα (label). Όλες οι εντολές με τα στοιχεία που ανέφερα αποθηκεύονται σε μια δομή δεδομένων τύπου δυναμικού πίνακα όπου κάθε γραμμή του πίνακα είναι μια νέα εντολή με τη σειρά που αναφέρεται στο πρόγραμμα. Η δομή αυτή ονομάζεται *"Instruction Data"*. Ο αναλυτής είναι αυτός που παρουσιάζει οποιαδήποτε συντακτικά λάθη εμφανίζονται στο πρόγραμμα και τερματίζει την εκτέλεσή του αν αυτό είναι απαραίτητο.

```
while(line!=NULL && internal_line<instruction_count)
{
    instructions[internal_line]=extract_instruction(line,labels,internal_line,actual_line);
    //the  use of strtok for diff strings changes functions internal state it mess the strings
    //substract line from text including /0 that strtok placed to tokenze from last point
    start_point+=strlen(line)+1;
    if(start_point<text_len)
        line=strtok(text+start_point,"\n");
    else
        line=NULL;
    PRINT_DEBUG_INFO(("Instruction %d Line: %s\n",internal_line,line));
    if(instructions[internal_line]!=NULL)
        internal_line++;
    actual_line++;
}
```

Σχήμα 4.14: Συντακτική Ανάλυση

Από αυτό το σημείο την εκτέλεση αναλαμβάνει ο πυρήνας του εξομοιωτή (core). Όταν ο πυρήνας δέχεται την εντολή κανονικής εκτέλεσης προχωράει στην εκτέλεση των εντολών που υπάρχουν μέσα στο *"Instruction Data"* μέχρι την εντολή *"STOP"*. Στην περίπτωση που ο πυρήνας δέχεται εντολή αποσφαλμάτωσης, τότε η

κάθε επόμενη εντολή βάσει του μετρητή προγράμματος "Programm Counter" εκτελείται κάθε φορά που λαμβάνει το *"Next Instruction"* από το γραφικό περιβάλλον.

Εδώ θέλω να αναφέρω ότι η διαχείριση των σημάτων από τον πυρήνα γίνεται με τον τρόπο του παραδείγματος που φαίνεται στο Σχήμα 4.15, δηλαδή για κάθε σήμα που ορίσαμε θα πρέπει να ορίσουμε και μια δράση όταν λαμβάνουμε το συγκεκριμένο σήμα. Όπως στο παράδειγμα όταν ενεργοποιείται το σήμα για άνοιγμα αρχείου κάνουμε τις σχετικές δράσεις, με το ίδιο τρόπο διαχειριζόμαστε και όλες τις εντολές όπως *"Next Instruction"*, *"Previous Instruction"*, "Execute" κ.ο.κ.

```
G_MODULE_EXPORT
void on_open_menu_item_activate(GtkMenuItem *menu_item,Simulator *sim
{
    gchar *filename;
    if (check_for_save(sim->editor)==TRUE)
    {
        on_save_menu_item_activate(NULL,sim);
    }
    filename=get_open_filename(sim);
    if(filename!=NULL) load_file_into_buffer(sim,filename);

}
```

Σχήμα 4.15: Διαχείριση Σημάτων

Όταν ο πυρήνας μπαίνει σε μια από τις δύο λειτουργίες, αρχίζει να εκτελεί εντολές. Η μονάδα που εκτελεί μια εντολή είναι το *"Instruction Execution"*. Υπάρχει ακόμα μια δομή δεδομένων που συνδέει το μοναδικό αριθμό μια εντολής του αντίστροφου προγράμματος με την αντίστοιχη λειτουργία της, δηλαδή με μια συνάρτηση. Η δομή αυτή ονομάζεται *"Instruction Function"*. Από το *"Instruction Function"* η μονάδα *"Instruction Execution"* εντοπίζει τη συνάρτηση που πρέπει να καλέσει ανάλογα με τον μοναδικό αριθμό μιας εντολής μέσα στον *"Instruction Data"*. Η επιλογή της εντολής μέσα στο *"Instruction Data"* γίνεται βάσει του μετρητή προγράμματος *"Program Counter"* και τέλος, γίνεται η εκτέλεση.

Οι εντολές που ανήκουν στην ίδια ομάδα από τις αριθμητικές εντολές, τις εντολές βρόγχων ή την εντολή μήνης έχουν παρόμοιο τρόπο εκτέλεσης για αυτό θα δώσω ένα παράδειγμα για κάθε ομάδα. Όσον αφορά τις αριθμητικές εντολές, στο

Σχήμα 4.16 δίνεται το παράδειγμα της εντολής “ADD”. Από εδώ βλέπουμε ότι σύμφωνα με τον ορισμό της εντολής από την αρχιτεκτονική Bob [21] κατά την εκτέλεση προς τα μπροστά εκτελείται η πρόσθεση και κατά την εκτέλεση προς τα πίσω η αφαίρεση. Οι πράξεις γίνονται πάνω στις τιμές των καταχωρητών που αποθηκεύονται στη δομή δεδομένων *“Registers”* από όπου διαβάζονται και γράφονται οι τιμές των καταχωρητών που έχει η κάθε εντολή σαν είσοδο. Η δομή αυτή είναι ένας πίνακας που χωράει δεκαέξι τιμές των δεκαέξι διφυών η κάθε μια.

```
void add_func(Instr_input input)
{
    if(simul->mv.direction=='F')
        simul->storage.REGISTERS[input.regd]+=simul->storage.REGISTERS[input.regs];
    else
        simul->storage.REGISTERS[input.regd]-=simul->storage.REGISTERS[input.regs];
    update_register_g(simul,input.regd,simul->storage.REGISTERS[input.regd]);
    update_pc(1);
}
```

Σχήμα 4.16: Εντολή Αθροίσματος

Οι εντολές βρόγχων ανανεώνουν τον καταχωρητή βρόγχων όπως ορίζεται στο Κεφάλαιο 3 εάν ισχύει η συνθήκη που ορίζει η κάθε εντολή όπως φαίνεται στα Σχήμα 4.17 και Σχήμα 4.18. Οι εντολές σε αυτή την ομάδα διαβάζουν τιμές από τη δομή *“Registers”* ώστε να εκτελέσουν τους ελέγχους τους.

```
void bgez_func(Instr_input input)
{
    if(simul->storage.REGISTERS[input.regd]>=0)
    {
        update_branch(input.off);
    }
    else
        update_pc(1);
}
```

Σχήμα 4.17: Εντολή Βρόγχου

```
void update_branch(short off)
{
    if(simul->mv.direction=='F')
        simul->mv.branch_register+=off;
    else
        simul->mv.branch_register-=off;
    update_pc(1);
}
```

Σχήμα 4.18: Αλλαγή Καταχωρητή Βρόγχων

Η τελευταία ομάδα εντολών αποτελείται από μια εντολή μνήμης που βλέπουμε στο Σχήμα 4.19. Η εντολή αντιμεταθέτει την τιμή της μνήμης -δομή δεδομένων *"Memory"*- στη διεύθυνση που κρατάει ο δεύτερος καταχωρητής, τον οποίο παίρνει σαν είσοδο, με την τιμή του πρώτου καταχωρητή, που παίρνει σαν είσοδο. Η δομή δεδομένων *"Memory"* είναι υλοποιημένη με αποδοτικό τρόπο ώστε ο εξομοιωτής να καταναλώνει τη μικρότερη δυνατή μνήμη, αλλά και να παρέχει πολύ γρήγορες προσβάσεις. Συγκεκριμένα, η μνήμη είναι ένας δισδιάστατος δυναμικός πίνακας που το μέγεθός του προσαρμόζεται ανάλογα με τον αριθμό των διευθύνσεων που χρησιμοποιούνται. Όσο πιο κοντινές η μία στην άλλη είναι οι διευθύνσεις, τόσο λιγότερη μνήμη θα δεσμεύσει ο εξομοιωτής. Η δομή αυτή συνοδεύεται από πράξεις εγγραφής και ανάγνωσης σε χρόνο O(1) εάν η διεύθυνση έχει ήδη δεσμευτεί από τον εξομοιωτή.

```
void exch_func(Instr_input input)
{
    short d=simul->storage.REGISTERS[input.regd];
    simul->storage.REGISTERS[input.regd]=read_from_mem(simul->storage.REGISTERS[input.regs]);
    write_to_mem(simul->storage.REGISTERS[input.regs],d);
    //update register on gui
    update_register_g(simul,input.regd,simul->storage.REGISTERS[input.regd]);
    update_pc(1);
}
```

Σχήμα 4.19: Εντολή Μνήμης

Με το τέλος της εκτέλεσης της κάθε εντολής η μονάδα *"Instruction Execution"* καλεί συναρτήσεις από την μονάδα *"GUI Builder"* ώστε να εμφανίσει τις αλλαγές

πάνω στο γραφικό περιβάλλον. Παραδείγματος χάρη, ο κώδικας που ανανεώνει τους καταχωρητές στο γραφικό περιβάλλον φαίνεται στο Σχήμα 4.20.

```
void update_register_g(Simulator *sim,short index,short val)
{
    char *str=(char*)malloc(6); //6 is the length of short number
    itoa_m(val,str);
    gtk_entry_set_text(sim->registers_tab.registers_g[index],str);
    free(str);
}
```

Σχήμα 4.20: Αλλαγή Καταχωρητή στην Διεπαφή

Η τελευταία δυνατότητα της μονάδας *"Instruction Execution"* είναι να υπολογίζει τον επόμενο μετρητή προγράμματος *"Program Counter"* προχωρώντας είτε στην επόμενη -ή προηγούμενη, ανάλογα της κατεύθυνσης- εντολή, είτε κάνοντας μια μετακίνηση μετά από εντολή βρόγχων. Ο κώδικας αυτός φαίνεται στο Σχήμα 4.21.

```
void update_pc(short off)
{
    simul->mv.prev_pc=simul->mv.program_counter;
    if(simul->mv.branch_register!=0)
    {
        if(simul->mv.direction=='F')
        {
            simul->mv.program_counter+=simul->mv.branch_register;
        }
        else
        {
            simul->mv.program_counter-=(simul->mv.branch_register);
        }
        update_pc_g(simul,simul->mv);
        return;
    }
    if(simul->mv.direction=='F')
        simul->mv.program_counter+=off;
    else
        simul->mv.program_counter-=off;
    update_pc_g(simul,simul->mv);
}
```

Σχήμα 4.21: Αλλαγή Μετρητή Προγράμματος

Η τελευταία μονάδα *"GUI Builder"* είναι ένα σύνολο συναρτήσεων που διαχειρίζονται το γραφικό περιβάλλον. Αρχικά, φορτώνει την περιγραφή του γραφικού περιβάλλοντος και παρουσιάζει το γραφικό περιβάλλον στην οθόνη με τον τρόπο που φαίνεται στο Σχήμα 4.22 όπου φορτώνουμε το κύριο παράθυρο, την γραμμή κατάστασης, και τα κουμπιά "Execute", "Debug" και "Stop" τα οποία κατά την διάρκεια λειτουργίας του εξομοιωτή θα αποστέλλουν σήματα στον πυρήνα, όπως τα εξηγήσαμε πριν.

```
sim->window = GTK_WIDGET (gtk_builder_get_object (sim->builder, "window"));
sim->status_bar=GTK_WIDGET(gtk_builder_get_object(sim->builder,"status_bar"));
sim->execute=GTK_BUTTON(gtk_builder_get_object(sim->builder,"execute_btn"));
sim->stop_btn=GTK_BUTTON(gtk_builder_get_object(sim->builder,"stop_exec_btn"));
sim->debug=GTK_CHECK_BUTTON(gtk_builder_get_object(sim->builder,"debug_check"));
gtk_builder_connect_signals (sim->builder,sim);
```

Σχήμα 4.22: Χτίσιμο Διεπαφής

4.6 Διαθέσιμα Συστήματα

Ο εξομοιωτής αναπτύχθηκε πάνω σε σύστημα Linux και επομένως μπορεί να χρησιμοποιηθεί σε οποιαδήποτε κατανομή – distribution – των Linux που έχουν υποστήριξη βιβλιοθηκών GTK ή QT. Επίσης, ο εξομοιωτής μπορεί να χρησιμοποιηθεί σε συστήματα Windows εγκαθιστώντας τις απαραίτητες βιβλιοθήκες και έχοντας το εκτελέσιμο αρχείο. Λεπτομέρειες εγκατάστασης καθώς και εκτελέσιμα αρχεία για όλα τα διαθέσιμα συστήματα μπορείτε να βρείτε στον ιστότοπο [24] του εξομοιωτή. Περισσότερες λεπτομέρειες για τον ιστότοπο δίνονται στο Κεφάλαιο 6.

Κεφάλαιο 5

Παραδείγματα Προγραμμάτων

5.1 Παραδείγματα Εκτέλεσης Προγραμμάτων

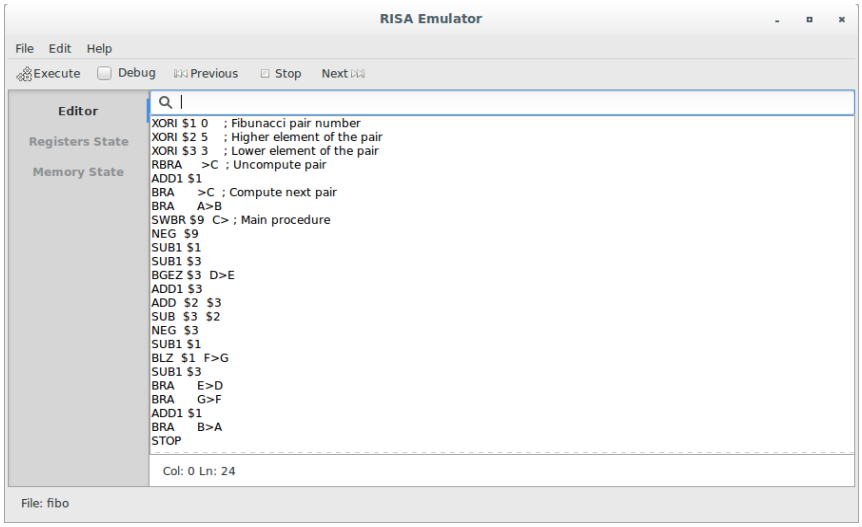
Σε αυτό το υποκεφάλαιο θα παρουσιάσω προγράμματα που έχουν γραφτεί με το σύνολο εντολών που παρουσιάστηκε στο Κεφάλαιο 3, την εκτέλεσή τους και την συλλογή αποτελεσμάτων.

5.1.1 Πρόγραμμα υπολογισμού σειράς Fibonacci

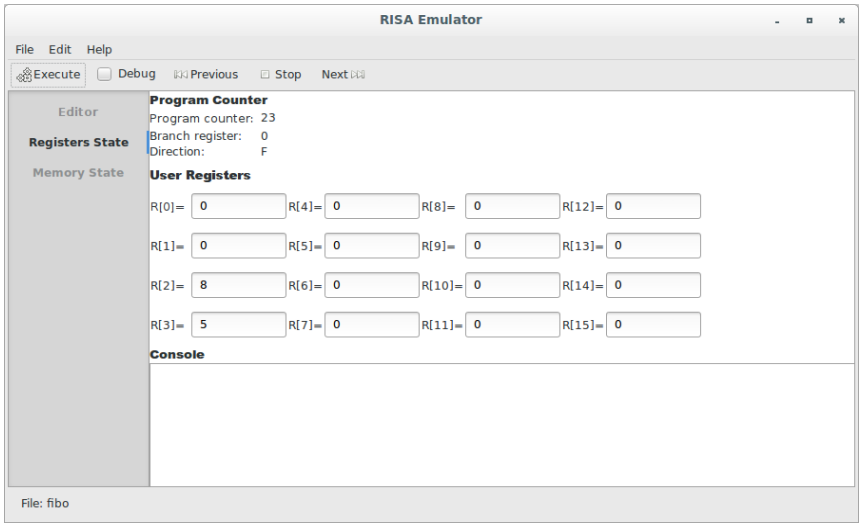
Ο σκοπός του συγκεκριμένου προγράμματος είναι ο εξής: το πρόγραμμα δέχεται ως είσοδο δύο αριθμούς της σειράς Fibonacci οι οποίοι φορτώνονται στους καταχωρητές 2 -ο μεγάλος αριθμός- και 3 -ο μικρότερος αριθμός. Το ίδιο το πρόγραμμα σκοπό έχει να υπολογίζει τα ζευγάρια της σειράς Fibonacci, δηλαδή δύο διαδοχικούς αριθμούς που ανήκουν σε αυτή την σειρά, αλλά για σκοπούς απλούστευσης θα υπολογίσουμε μέχρι το επόμενο ζευγάρι της σειράς. Ο επιπλέον υπολογισμός που κάνει το πρόγραμμα είναι με τον ίδιο κώδικα που υπολογίζει τα επόμενα ζευγάρια Fibonacci να υπολογίσει και τα προηγούμενα εκτελώντας τον κώδικα με προς τα πίσω κατεύθυνση. Αυτό θα φανεί καλύτερα στο παράδειγμα αποσφαλμάτωσης.

Ο κώδικας του εν λόγω προγράμματος φαίνεται στο Σχήμα 5.1 και τα αποτελέσματα μετά την εκτέλεση στο Σχήμα 5.2. Στο τελευταίο σχήμα μπορούμε να δούμε τις καταστάσεις των καταχωρητών με το τέλος της εκτέλεσης του προγράμματος

και να επιβεβαιώσουμε πως όντως οι καταχωρητές 2 και 3 έχουν τις τιμές του επόμενου ζευγαριού Fibonacci από αυτό που δώσαμε ως είσοδο.



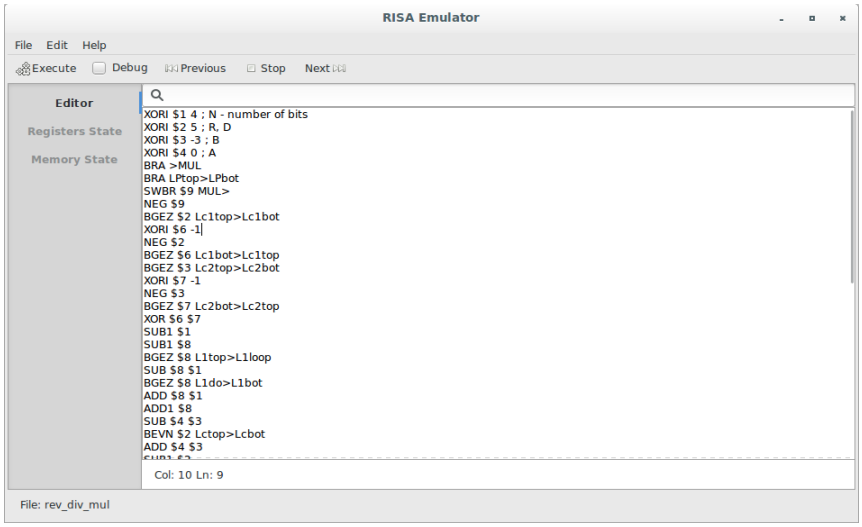
Σχήμα 5.1: Παράδειγμα Εκτέλεσης Fibonacci



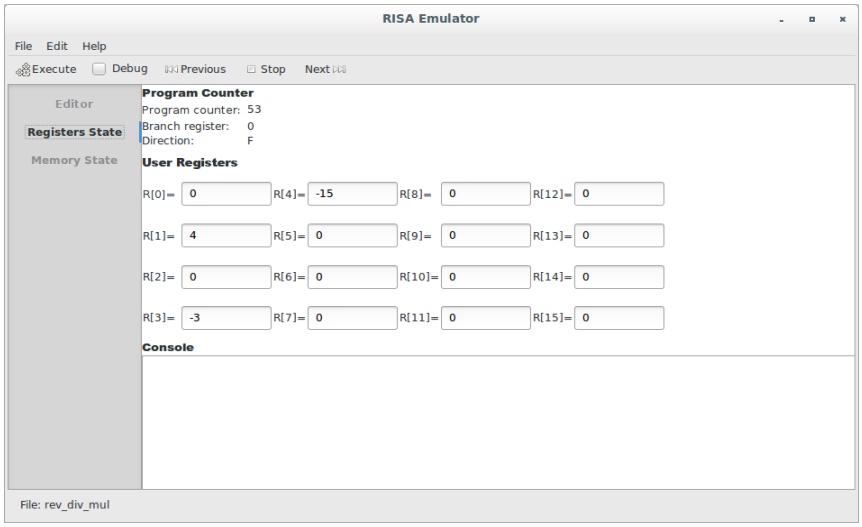
Σχήμα 5.2: Αποτέλεσμα Εκτέλεσης Fibonacci

5.1.2 Πρόγραμμα υπολογισμού γινόμενου

Σε αυτό το παράδειγμα θα δούμε το πρόγραμμα που υπολογίζει το γινόμενο δύο αριθμών, καθώς δεν υπάρχει απευθείας εντολή πολλαπλασιασμού στο σύνολο εντολών που υποστηρίζει η συγκεκριμένη αρχιτεκτονική.



Σχήμα 5.3: Παράδειγμα Εκτέλεσης Πολλαπλασιασμού



Σχήμα 5.4: Αποτέλεσμα Εκτέλεσης Πολλαπλασιασμού

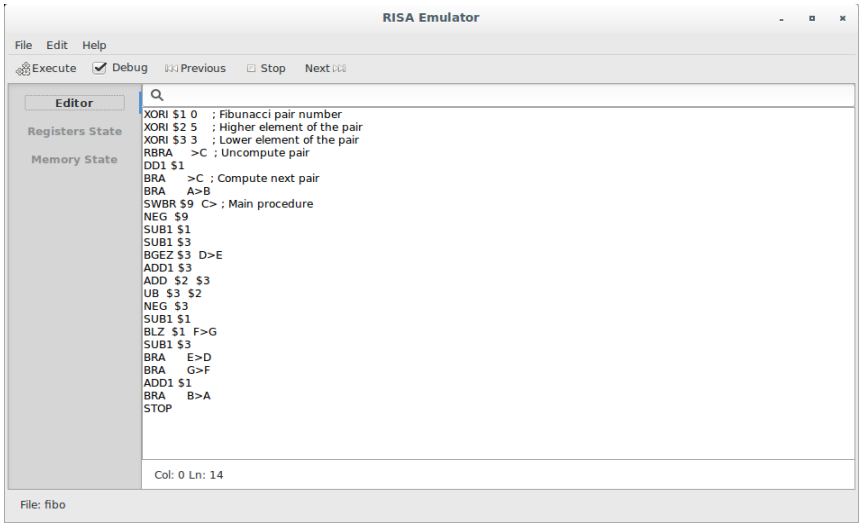
Στο Σχήμα 5.3 βλέπουμε ένα κομμάτι κώδικα από το πρόγραμμα πολλαπλασιασμού. Στους καταχωρητές 2 και 3 φορτώνουμε τους αριθμούς που θέλουμε να πολλαπλασιάσουμε.

Μέτα την εκτέλεση του προγράμματος μπορούμε στο Σχήμα 5.4 να δούμε τις καταστάσεις των καταχωρητών. Ο καταχωρητής 4 όντως περιέχει το γινόμενο των αριθμών εισόδου που δόθηκαν.

Περισσότερα παραδείγματα προγραμμάτων υπάρχουν στον ιστότοπο του εξομοιωτή που θα παρουσιάσω στο τελευταίο κεφάλαιο.

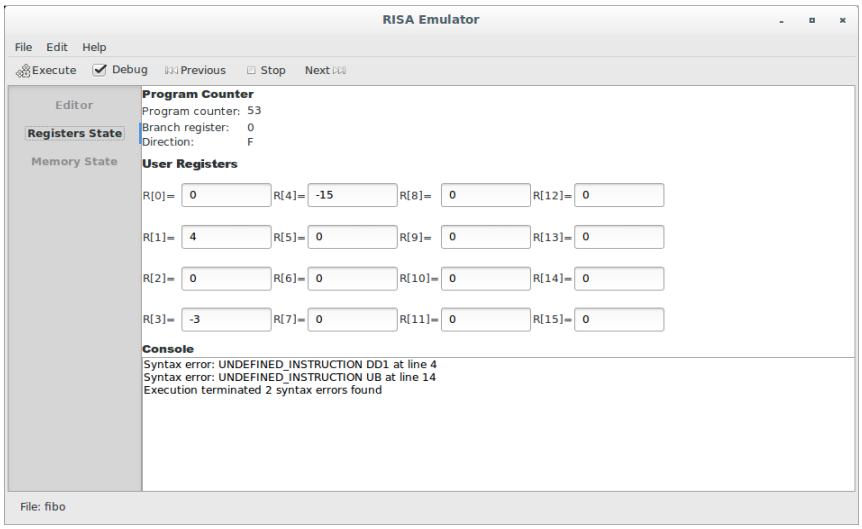
5.2 Παράδειγμα αποσφαλμάτωσης προγράμματος

Σε αυτό το υποκεφάλαιο θα υποθέσουμε ότι το πρόγραμμα υπολογισμού ζευγαριών Fibonacci έχει συντακτικά και λογικά λάθη. Παρατηρούμε ότι στις γραμμές 4 και 14 στον Σχήμα 5.5 υπάρχουν συντακτικά λάθη.



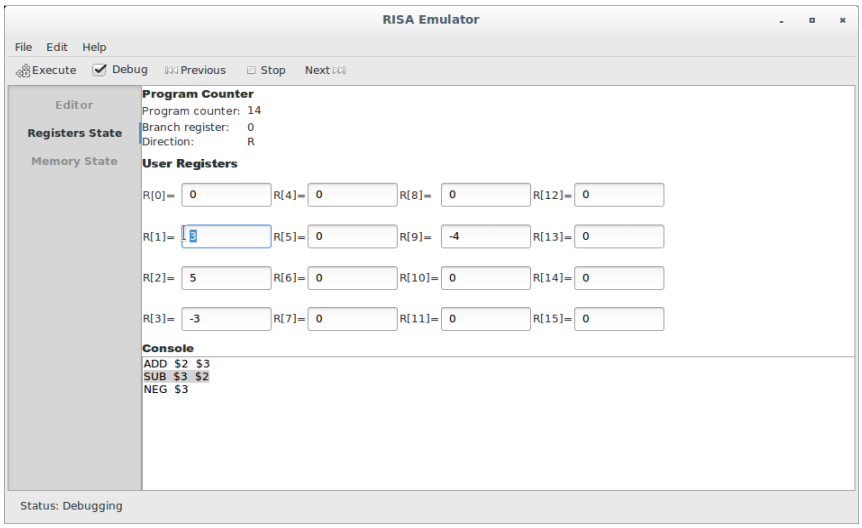
Σχήμα 5.5: Παράδειγμα Αποσφαλμάτωσης Fibonacci

Ο εξομοιωτής επισημαίνει τα συντακτικά λάθη πάνω στην κονσόλα, όπως βλέπουμε στο Σχήμα 5.6.



Σχήμα 5.6: Συντακτικά λάθη Fibonacci.

Προχωρώντας στην διαδικασία αποσφαλμάτωσης -μπροστά και πίσω, αν χρειαστεί- ας υποθέσουμε ότι υπάρχει ένα λογικό λάθος και η τιμή σε έναν καταχωρητή δεν είναι η αναμενόμενη για την συνέχεια. Διορθώνουμε την τιμή αυτή όπως φαίνεται στο Σχήμα 5.7 ώστε να μην τερματίσουμε την αποσφαλμάτωση και έτσι ολοκληρώνουμε τη διαδικασία φτάνοντας σε αναμενόμενα αποτελέσματα.



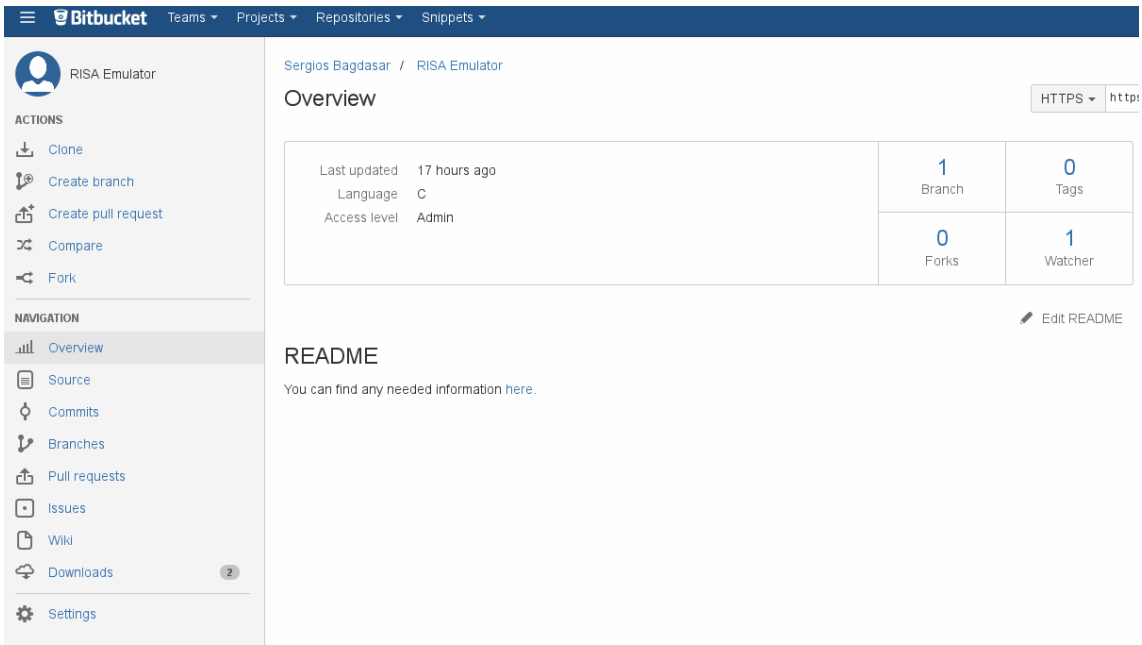
Σχήμα 5.7: Διόρθωση Καταχωρητή.

Κεφάλαιο 6

Ιστότοπος

6.1 Οργάνωση

Σε αυτό το κεφάλαιο γίνεται αναφορά στον ιστότοπο του εργαλείου. Ο ιστότοπος του εργαλείου φιλοξενείται από την εταιρεία BitBucket. Εκεί μπορείτε να βρείτε τον πηγαίο κώδικα του εργαλείου, καθώς και όλες τις απαραίτητες πληροφορίες και οδηγίες για την εγκατάσταση και χρήση του εξομοιωτή. Μπορείτε να βρείτε τον ιστότοπο στην ηλεκτρονική διεύθυνση που υπάρχει στην βιβλιογραφία [25]. Πιο κάτω, γίνεται μια λεπτομερείς παρουσίαση των τρόπων αξιοποίησης του ιστότοπου βασισμένη στο Σχήμα 6.1 που είναι η αρχική σελίδα του ιστότοπου, όπου μπορούμε να δούμε το μενού περιήγησης.



Σχήμα 6.1: Πρόσοψη Ιστότοπου

6.2 Πηγαίος κώδικας

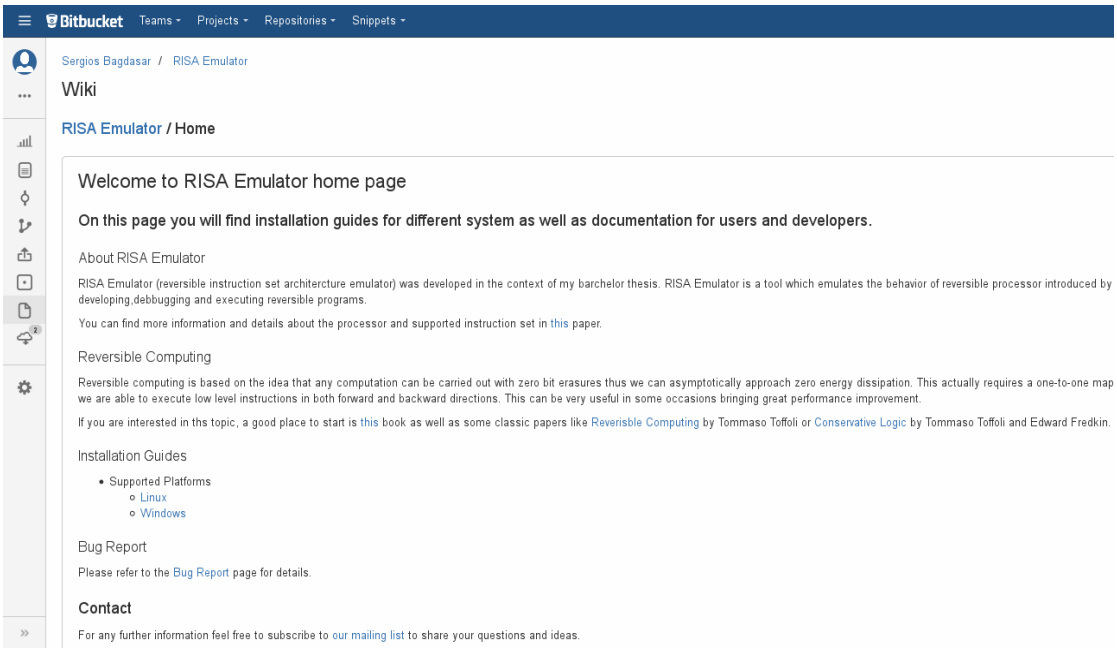
Μπορείτε να βρείτε τον πηγαίο κώδικα του εργαλείου πατώντας στο *"Source"* όπως φαίνεται στο Σχήμα 6.1. Στο παράθυρο που θα εμφανιστεί μπορείτε να βρείτε όλους τους καταλόγους που περιλαμβάνουν, εκτός από τον πηγαίο κώδικα και το αρχείο περιγραφής του γραφικού περιβάλλοντος και παραδείγματα προγραμμάτων που μπορείτε να δοκιμάσετε.

6.3 Αναφορά προβλημάτων και προτάσεις

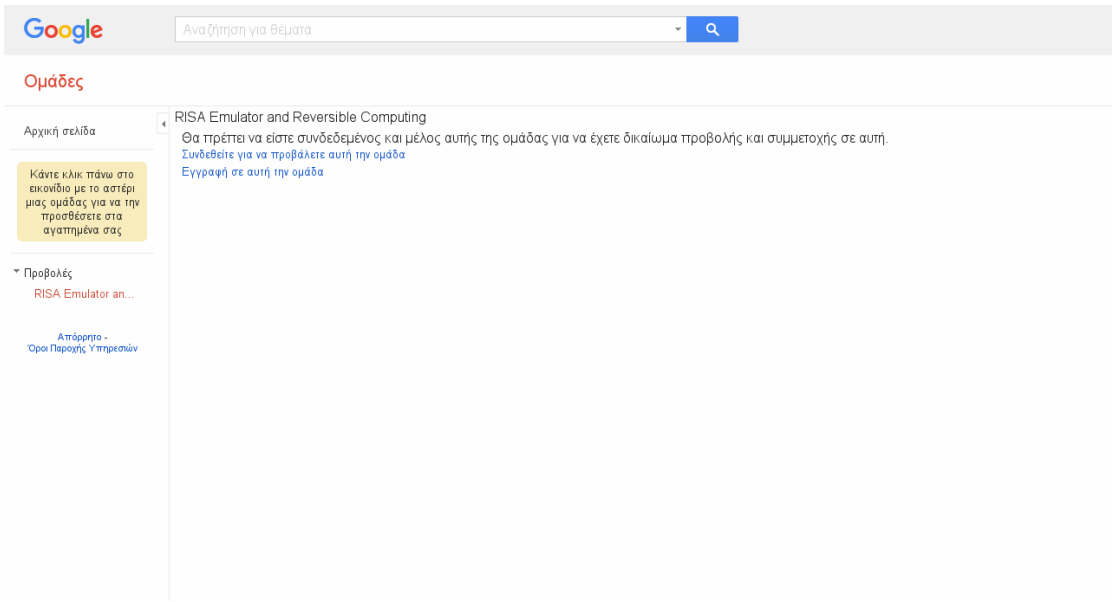
Μπορείτε να βρείτε το παράθυρο αναφοράς προβλημάτων του εργαλείου πατώντας στο *"Issues"*, όπως φαίνεται στο Σχήμα 6.1. Σε αυτό το παράθυρο μπορείτε να αναφέρεται τυχόν προβλήματα που παρουσιάστηκαν κατά τη χρήση του εργαλείου ή να προτείνετε νέες λειτουργίες ή/και παρατηρήσεις.

6.4 Σελίδα Wiki και τρόποι εμπλοκής

Στο Σχήμα 6.2 βλέπουμε την σελίδα Wiki του εργαλείου, η οποία περιέχει πληροφορίες για χρήστες του εργαλείου, για προγραμματιστές που θέλουν να συνεισφέρουν στην ανάπτυξή του, καθώς και για νέους ερευνητές στο πεδίο του αντίστροφου υπολογισμού. Επίσης, υπάρχει μια σελίδα με ιδέες για μελλοντική εργασία, που αναφέρονται και στο τελευταίο κεφάλαιο. Τέλος, μπορείτε εύκολα να επικοινωνήσετε με τους υπόλοιπους προγραμματιστές κάνοντας εγγραφή στη λίστα ηλεκτρονικών διευθύνσεων -mailing list- και ταυτόχρονα στην ομάδα συζήτησης που έχει δημιουργηθεί, όπως μπορείτε να δείτε στο Σχήμα 6.3.



Σχήμα 6.2: Σελίδα Wiki



Σχήμα 6.3: Ομάδα συζήτησης

Κεφάλαιο 7

Συμπεράσματα και Μελλοντική Εργασία

7.1 Συμπεράσματα

Φτάνοντας στην ολοκλήρωση της διπλωματικής αυτή εργασίας αναπτύξαμε ένα εξομοιωτή εντολών αντίστροφης αρχιτεκτονικής. Ο εξομοιωτής μας επιτρέπει την ανάπτυξη την αποσφαλμάτωση και την εκτέλεση αντίστροφων προγραμμάτων. Είναι ανοιχτού πηγαίου κώδικα και άρα μπορεί η ανάπτυξή του να συνεχιστεί συλλογικά και επίσης μπορεί να χρησιμοποιηθεί ως εργαλείο για διεκπεραίωση άλλων ερευνών. Τέλος είναι διαθέσιμος για διάφορα λειτουργικά συστήματα ώστε να στοχεύσει σε περισσότερους χρήστες.

7.2 Συντακτικός Αναλυτής

Παρ' όλη τη δουλειά, ο εξομοιωτής χρειάζεται ακόμα μερικές βελτιώσεις. Μία από αυτές είναι η βελτίωση του συντακτικού αναλυτή με ειδικά εργαλεία ανάπτυξης μεταγλωττιστών, ώστε να γίνεται ευκολότερη επέκταση του συνόλου εντολών που υποστηρίζει, καθώς και αποτελεσματικότερη ανάλυση συντακτικών λαθών. Η βελτίωση του συντακτικού αναλυτή θα διευκολύνει την ανάπτυξη πιο σύνθετων προγραμμάτων.

7.3 Συγγραφή και δοκιμή προγραμμάτων

Ο εξομοιωτής δίνει μια νέα ευκαιρία να αναπτύξουμε αντίστροφα προγράμματα. Το αρχικό κίνητρο για την ανάπτυξη του εξομοιωτή ήταν, πέρα από το ακαδημαϊκό ενδιαφέρον, η ανάγκη να δείξουμε να πραγματικά οφέλη και δυνατότητες του αντίστροφου υπολογισμού σε επίπεδο προγραμμάτων. Χρειάζονται, λοιπόν, να προστεθούν δυνατότητες μέτρησης επιδόσεων και σύγκρισης των αντίστροφων και μη προγραμμάτων, αλλά και συγγραφή νέων, πιο σύνθετων προγραμμάτων που θα μπορούσαν να έχουν πραγματική χρήση. Για να επιτευχθεί όμως αυτό χρειάζεται και εμπλουτισμός του συνόλου εντολών της αντίστροφης αρχιτεκτονικής που εξομοιώνουμε με περισσότερες εντολές που υποστηρίζουν σύγχρονοι επεξεργαστές.

7.4 Μεταγλωττιστής

Θα μπορούσαμε να φτιάξουμε έναν μεταγλωττιστή για μια απλή γλώσσα προγραμματίσου, όπως η γλώσσα C- [27] -που είναι μια περιορισμένη γλώσσα C -, με επιπλέον αντίστροφες λειτουργίες που να παράγει πρόγραμμα στην συμβολική γλώσσα προγραμματισμού που μπορεί να εκτελέσει ο εξομοιωτής. Έτσι θα γινόταν πιο εύκολο να φτιάξουμε πιο πολύπλοκα προγράμματα και η νέα γλώσσα θα ήταν πολύ πιο ελκυστική για νέους ερευνητές στον πεδίο.

7.5 Ανοιχτός πηγαίος κώδικας

Πέρα από όσα αναφέρονται σε αυτή την διπλωματική εργασία, ο αντίστροφος υπολογισμός χρειάζεται έρευνα σε πολλούς άλλους τομείς. Τα κίνητρα είναι δεδομένα, αλλά θα πρέπει να εμπλακούν και άλλοι ερευνητές ώστε να υπάρξει ικανοποιητική πρόοδος. Κατά την άποψη μου, ο μόνος τρόπος για να γίνει αυτό είναι η δημοσιοποίηση της δουλείας που γίνεται, ώστε ο καθένας να έχει μια βάση να συνεχίσει την έρευνα. Θεωρώ πως σε αυτή τη φάση προέχει η ανάγκη να κάνουμε ένα μεγάλο βήμα στην επιστήμη μας, κάτι που μπορούμε να το πετύχουμε δουλεύοντας συντονισμένα και με αυτό ολοκληρώνω τη διπλωματική μου εργασία.

Βιβλιογραφία

[1] T. Toffoli, “Reversible Computing,” in Proceedings of the 7th Colloquium on Automata, Languages and Programming, 1980, pp. 632–644.

[2] K. S. Perumalla, Introduction to Reversible Computing, 1st ed. Chapman & Hall/CRC, 2013.

[3] K. Salah, Y. Ismail, and A. El-Rouby, “Introduction: Work Around Moore’s Law,” *Arbitr. Model. TSVs 3D Integr. Circuits*, pp. 1–15, 2015.

[4] R. Landauer, “Irreversibility and Heat Generation in the Computing Process,” IBM J. Res. Dev., vol. 5, no. July, pp. 183–191, 1961.

[5] J. Ren and V. K. Semenov, “Progress With Physically and Logically Reversible Superconducting Digital Circuits,” *IEEE Trans. Appl. Supercond.*, vol. 21, no. 3, pp. 780–786, 2011.

[6] M. Dubois, M. Annavaram, and P. Stenstrm, Parallel Computer Organization and Design. New York, NY, USA: Cambridge University Press, 2012.

[7] M. Bahi and C. Eisenbeis, “Rematerialization-based Register Allocation Through Reverse Computing,” in Proceedings of the 8th ACM International Conference on Computing Frontiers, 2011, pp. 24:1–24:2.

[8] E. Fredkin and T. Toffoli, “Conservative logic,” *Int. J. Theor. Phys.*, vol. 21, no. 3–4, pp. 219–253, 1982.

[9] K. Morita and Y. Yamaguchi, “A universal reversible turing machine,” *5th Int. Conf. Mach. Comput. Universality, MCU 2007*, vol. 4664 LNCS, pp. 90–98, 2007.

[10] C. H. Bennett, “The thermodynamics of computation—a review,” *Int. J. Theor. Phys.*, vol. 21, no. 12, pp. 905–940, Dec. 1982.

[11] M. Soeken, S. Frehse, R. Wille, and R. Drechsler, “RevKit : A Toolkit for Reversible Circuit Design,” Mult. Log. Soft Comput., pp. 10–13, 2012.

[12] R. Wille and R. Drechsler, “BDD-based synthesis of reversible logic for large functions,” in Proceedings of the 46th Annual Design Automation Conference, 2009, p. 270.

[13] [1] D. M. Miller, D. Maslov, G. W. Dueck, and A. C. M. Acm, “A transformation based algorithm for reversible logic synthesis,” in 40th Design Automation Conference, Proceedings 2003, 2003, pp. 318–323.

[14] M. Soeken, R. Wille, and R. Drechsler, “Hierarchical synthesis of reversible circuits using positive and negative Davio decomposition,” in IDT’10 - 2010 5th International Design and Test Workshop, Proceedings, 2010, pp. 143–148.

[15] M. Soeken, R. Wille, G. W. Dueck, and R. Drechsler, “Window optimization of reversible and quantum circuits,” in Proceedings of the 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2010, 2010, pp. 341–345.

[16] R. Wille, M. Soeken, and R. Drechsler, “Reducing the number of lines in reversible circuits,” in Design Automation Conference (DAC), 2010 47th ACM/IEEE, 2010, pp. 647–652.

[17] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, “RevLib: An online resource for reversible functions and reversible circuits,” in Proceedings of The International Symposium on Multiple-Valued Logic, 2008, pp. 220–225.

[18] R. Wille, S. Offermann, and R. Drechsler, SyReC: A programming language for synthesis of reversible circuits, Specifications, vol. 106 LNEE. Springer New York, 2012.

[19] J. Storrs Hall, “A reversible instruction set architecture and algorithms,” in Proceedings Workshop on Physics and Computation. PhysComp ’94, 1994, pp. 128–134.

[20] C. J. Vieri, “Pendulum: A reversible computer architecture,” Technology, no. 1993, 1995.

[21] M. K. Thomsen, H. B. Axelsen, and R. Glück, “A Reversible Processor Architecture and its Reversible Logic Design,” 3nd Work. Revers. Comput. Proc., pp. 123–134, 2011.

[22] R. Wille, M. Soeken, D. Große, E. Schönborn, and R. Drechsler, “Designing a RISC CPU in reversible logic,” *Proc. - 41st IEEE Int. Symp. Mult. Logic, ISMVL 2011*, pp. 170–175, 2011.

[23] C. Lutz and H. Derby, “Janus: a time-reversible language,” Caltech Class Project, 1982.

[24] Bagdasar Sergiy, “RISA Emulator Wiki.” [Online]. Available: https://bitbucket.org/sbagda01/risa-emulator/wiki/Home.

[25] Bagdasar Sergiy, “RISA Emulator,” 2016. [Online]. Available: https://bitbucket.org/sbagda01/risa-emulator/overview.

[26] GNOME Developers, “Glade- A User Interface Designer.” [Online]. Available: https://glade.gnome.org/.

[27] “C-Minus Programming Language Specifications.” [Online]. Available: http://www.cs.dartmouth.edu/~cs57/Project/C- Spec.pdf.