Διεθνές Πανεπιστήμιο Ελλάδος
Τμήμα Μηχανικών Πληροφορικής , Υπολογιστών και Τηλεπικοινωνιών

Υλοποίηση συμβολομεταφραστή (assembler) για τον επεξεργαστή Robin

Πτυχιακή Εργασία

Ζουμπαρά Ειρήνη ΑΜ : 3802

Επιβλέπων: Ιωάννης Καλόμοιρος, Καθηγητής

Σέρρες, Μάρτιος 2021

Υπεύθυνη Δήλωση : Βεβαιώνω ότι είμαι η συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται επακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής, Υπολογιστών και τηλεπικοινωνιών του Διεθνούς Πανεπιστημίου Ελλάδας.

Περίληψη

Ο σκοπός της παρούσης πτυχιακής εργασίας είναι η υλοποίηση ενός συμβολομεταφραστή (assembler) για τον επεξεργαστή Robin. Ο επεξεργαστής Robin είναι ένας δοκιμαστικός εκπαιδευτικός επεξεργαστής, γραμμένος σε γλώσσα περιγραφής υλικού, ο οποίος υλοποιείται στο πλαίσιο της εκπαιδευτικής διαδικασίας, στο μάθημα του Τμήματος με τίτλο «Προηγμένα Ψηφιακά Συστήματα».

Ο συμβολομεταφραστής που υλοποιήθηκε στην παρούσα εργασία, μετατρέπει σε δυαδική μορφή τις εντολές που γράφονται από το χρήστη, ακολουθώντας τις προδιαγραφές και την εσωτερική δομή του επεξεργαστή Robin. Αρχικά, μελετήθηκαν οι προδιαγραφές του επεξεργαστή και το σύνολο των εντολών του. Η υλοποίηση της εφαρμογής το συμβολομεταφραστή έγινε με τη χρήση της γλώσσας προγραμματισμού Python και πραγματοποιήθηκε σε γραφικό περιβάλλον tkinter, ώστε η χρήση του να είναι φιλική προς τον χρήστη. Το αποτέλεσμα είναι μια γραφική διεπαφή (GUI) στην οποία γίνεται η εισαγωγή ενός αρχείου στην assembly του επεξεργαστή Robin και στη συνέχεια, μέσω των επιλογών του χρήστη από το μενού, υλοποιείται η μετάφραση του αρχείου σε δυαδική μορφή και η αποθήκευση σε ένα τελικό αρχείο κατάλληλο για φόρτωση στον επεξεργαστή.

Περιεχόμενα

Εισαγωγή 5
Κεφάλαιο 1 6
1.1 Ποιος είναι ο επεξεργαστής Robin και ποιες οι βασικές προδιαγραφές του 7
1.2 Η Αρχιτεκτονική του επεξεργαστή Robin 8
1.3 Instruction word 11
1.4 Register file 14
1.5 Περιγραφή των καταχωρητών του συστήματος 15
1.6 Σύνολο Εντολών 17
Κεφάλαιο 2 20
2.1 Γλώσσα προγραμματισμού ανάπτυξης κώδικα 21
2.2 Περιβάλλον ανάπτυξης κώδικα 21
2.3 Τύποι εντολών 21
2.4 Mnemonic – Opcode 22
2.5 Είδη ορισμάτων που μπορεί να πάρει το πρώτο operand 23
2.6 Κατηγορίες εντολών που απαιτούν συγκεκριμένους τύπους πρώτου operand 25
2.7 Είδη ορισμάτων που μπορεί να πάρει το δεύτερο operand 26
2.8 Κατηγορία εντολών που απαιτεί συγκεκριμένο τύπο δεύτερου operand 26
2.9 Εντολές που έχουν shamt 27
2.10 Function bit 27
2.11 Σύνταξη των εντολών για τον assembler 28
Κεφάλαιο 3 29
Τεχνικές ανάπτυξης κώδικα του συμβολομεταφραστή 29
3.1 Διάγραμμα Ροής του προγράμματος συμβολομεταφραστή 30
3.2 Η δομή του προγράμματος για το συμβολομεταφραστή (assembler) 32
3.2.1 Βιβλιοθήκες 32
3.2.2 Λεξικό 32
3.2.3 Βασικές μέθοδοι και συναρτήσεις 34
3.2.4 Επεξήγηση του κώδικα Assembler 37
3.2.5 Κυρίως πρόγραμμα και μετατροπή δεδομένων 40
3.3 Γραφικό περιβάλλον 53
3.3.1 Επιλογή framework για το γραφικό περιβάλλον (GUI) 53
3.3.2 Εισαγωγή λειτουργικής μονάδας και βιβλιοθήκης για το γραφικό περιβάλλον 53
3.3.3 Προσθήκη από βιβλιοθήκες και άλλα αρχεία 54

3.3.4 Κλάση και αντικείμενα 55
3.3.5 Επεξήγηση του κώδικα για την δημιουργία του γραφικού περιβάλλοντος (GUI) . 55
3.3.6 Δηλώσεις των μενού του αρχείου κειμένου 57
3.3.7 Δημιουργία συναρτήσεων για τη λειτουργία των μενού 59
3.3.8 Αρχείο Εξόδου 62
3.3.9 Αρχείο Σφαλμάτων 63
3.4 Παρουσίαση γραφικού περιβάλλοντος (GUI) 64
Κεφάλαιο 4 67
Αποτελέσματα της εργασίας και Εγχειρίδιο χρήσης Text Editor 67
Κεφάλαιο 5 80
5.1 Συμπεράσματα και προτάσεις βελτίωσης 80
5.2 Προτάσεις για μελλοντική έρευνα 80
Παράρτημα Α΄ 81
Κώδικας Assembler 81
Παράρτημα Β΄ 97
Κώδικας για τον Text Editor 97
Πρόγραμμα Εμφάνισης Αρχείου Εξόδου 102
Πρόγραμμα Εμφάνισης Σφαλμάτων 103
Βιβλιογραφία 104

Εισαγωγή

Ο επεξεργαστής Robin είναι ένας εκπαιδευτικός επεξεργαστής, ο οποίος δημιουργήθηκε από το Εργαστήριο Ρομποτικής και Ευφυών συστημάτων του Τμήματος Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών, του Διεθνούς Πανεπιστημίου Ελλάδος. Ο επεξεργαστής αναπτύσσεται σε γλώσσα περιγραφής υλικού VHDL, στο πλαίσιο σχεδίων εργασίας. Προς το παρόν, έχει αναπτυχθεί το μεγαλύτερο μέρος του πυρήνα του επεξεργαστή και έχει ελεγχθεί η λειτουργία του σε προσομοίωση. Ο σκοπός της δημιουργίας του είναι η χρήση του σαν διδακτικό υλικό για τις ανάγκες του Τμήματος. Συγκεκριμένα, μπορεί να χρησιμοποιηθεί ως μελέτη περίπτωσης (Case Study) στα μαθήματα «Αρχιτεκτονική Υπολογιστών» και «Προηγμένα Ψηφιακά Συστήματα», καθώς και ως πεδίο ανάπτυξης πτυχιακών και διπλωματικών εργασιών.

Όταν άρχισε η παρούσα πτυχιακή εργασία, ο επεξεργαστής προγραμματιζόταν σε γλώσσα μηχανής, με τη βοήθεια αρχείου δυαδικών εντολών, που διαβαζόταν από τη γλώσσα VHDL. Στη συγκεκριμένη εργασία, υλοποιείται ένας συμβολομεταφραστής για τον επεξεργαστή Robin. Ο σκοπός του συμβολομεταφραστή είναι να μετατρέπει τον κώδικα της assembly σε δυαδική μορφή. Έτσι, με βάση το αποτέλεσμα της παρούσας εργασίας, ο χρήστης μπορεί να γράφει κώδικα σε γλώσσα assembly, η οποία θα μετατρέπεται σε δυαδικό αρχείο γλώσσας μηχανής, προς φόρτωση στον επεξεργαστή. Η γλώσσα assembly που χρησιμοποιούμε στην παρούσα εργασία είναι αυτή που προβλέπεται από τις προδιαγραφές του επεξεργαστή, όπως παραδόθηκαν από τον επιβλέποντα καθηγητή.

Η παρούσα διπλωματική εργασία είναι ταυτόχρονα ένα εγχειρίδιο, το οποίο περιέχει οδηγίες για την συγγραφή του κώδικα για τον επεξεργαστή και καθοδήγηση για την χρήση του γραφικού περιβάλλοντος. Για τη συγγραφή του κώδικα σε μορφή assembly διευκρινίζονται τα πλαίσια και οι περιορισμοί, έτσι ώστε να αποφευχθούν τυχόν σφάλματα κατά την μετατροπή του κώδικα. Στη συνέχεια, γίνεται η παρουσίαση και η περιγραφή του γραφικού περιβάλλοντος.

Η πτυχιακή εργασία απαρτίζεται από τέσσερα κεφάλαια. Στο πρώτο γίνεται μία εισαγωγή για τον επεξεργαστή Robin, για τα μέρη από τα οποία αποτελείται καθώς και για την αρχιτεκτονική του. Στο δεύτερο κεφάλαιο γίνεται αναφορά στα εργαλεία προγραμματισμού και στις μεθόδους που χρησιμοποιήθηκαν. Στο τρίτο κεφάλαιο γίνεται παρουσίαση του κώδικα, ο οποίος αναπτύχθηκε για τον συμβολομεταφραστή και για το γραφικό περιβάλλον. Τέλος, στο τέταρτο κεφάλαιο αναφέρονται τα ερευνητικά πορίσματα της εργασίας καθώς και τρόποι βελτίωσης της στο μέλλον.

Κεφάλαιο 1

Προδιαγραφές και αρχιτεκτονική του επεξεργαστή Robin

Στο πρώτο κεφάλαιο γίνεται η παρουσίαση του επεξεργαστή Robin και των γενικών χαρακτηριστικών του. Στη συνέχεια, πραγματοποιείται αναλυτική περιγραφή των μερών από τα οποία αποτελείται και της αρχιτεκτονικής του. Επίσης, θα γίνει σχετική αναφορά στο Αρχείο Καταχωρητών (Register File) του συστήματος. Τέλος, θα αναλυθεί το σύνολο των εντολών (Instruction Set) του επεξεργαστή Robin.

1.1 Ποιος είναι ο επεξεργαστής Robin και ποιες οι βασικές προδιαγραφές του

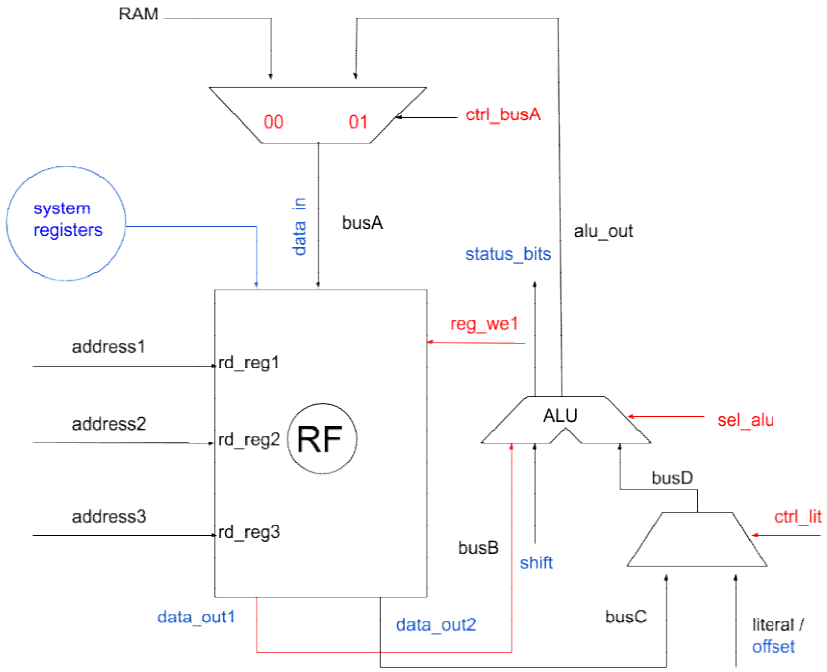
Ο επεξεργαστής Robin είναι ένας εκπαιδευτικός επεξεργαστής, ο οποίος είναι σχεδιασμένος με τη χρήση της γλώσσας περιγραφής υλικού (Hardware Description Language)VHDL. Ο συγκεκριμένος επεξεργαστής είναι σχεδιασμένος ως "System on Chip " (SoC). Με τον όρο αυτό χαρακτηρίζεται ένα σύστημα το οποίο εμπεριέχει έναν μικροεπεξεργαστή με τα βασικά περιφερειακά του σε ένα μόνο ολοκληρωμένο κύκλωμα. Το έργο αυτό αναπτύχθηκε από το Εργαστήριο Ρομποτικής και Ευφυών συστημάτων του Διεθνούς Πανεπιστημίου Ελλάδος, Πανεπιστημιούπολη Σερρών, με υπεύθυνο τον Καθηγητή Ιωάννη Καλόμοιρο.

Η έκδοση v0 (έκδοση μικροελεγκτή) του επεξεργαστή έχει σχεδιαστεί ως μικροελεγκτής ειδικού σκοπού 8 bit. Είναι επεξεργαστής RISC (Reduced Instruction Set Computer), όμως ακολουθεί και ορισμένες από τις αρχές των επεξεργαστών MIPS. Οι RISC εντολές είναι όλες σύντομες και έχουν όλες το ίδιο μήκος. Επίσης, ο επεξεργαστής Robin ακολουθεί ορισμένες από τις αρχές της Αρχιτεκτονικής Συνόλου Εντολών (Instruction Set Architecture) που εφαρμόζονται στους μικροελεγκτές Microchip PIC.

Ως εκπαιδευτικός επεξεργαστής, ο Robin System on Chip στη βασική του έκδοση υλοποιεί ένα πολύ μειωμένο σύνολο εντολών (Very-RISC ή v-RISC, κατά την ορολογία που υιοθετήσαμε), με βασικές εντολές τύπου "move" και τύπου αριθμητικής επεξεργασίας. Διαθέτει στοίβα 8 επιπέδων για εμφωλευμένες κλήσεις υπορουτινών. Ο συγκεκριμένος επεξεργαστής προορίζεται με σκοπό την υλοποίηση διαμόρφωσης μιας διάταξης MAX10 χαμηλού κόστους, από την Intel.
Η τεκμηρίωση του επεξεργαστή Robin θα αποτελέσει μέρος του διδακτικού υλικού σε μαθήματα σχεδίασης ψηφιακών συστημάτων και αρχιτεκτονικής υπολογιστών. Δεδομένου ότι αποτελεί έναν εισαγωγικό και εύκολο στην κατανόηση, αλλά ταυτόχρονα, ισχυρό επεξεργαστή, η διδασκαλία του Robin θα αξιολογηθεί και ως εκπαιδευτικό παράδειγμα. Παρακάτω, παρουσιάζουμε τις βασικές προδιαγραφές του συστήματος, όπως αυτές έχουν ληφθεί από την τεκμηρίωση του επεξεργαστή, που είναι σε εξέλιξη (βλέπε [1]: Ioannis Kalomiros, The Robin Processor, MCU Edition: Basic Specifications, v.0, 2019, IHU, Serres Campus).

1.2 Η Αρχιτεκτονική του επεξεργαστή Robin

Ο κύριος πυρήνας του επεξεργαστή Robin αποτελείται από τον αποκωδικοποιητή εντολών, ένα αρχείο καταχωρητών (register file-RF), μια ALU και μια μονάδα ελέγχου. Ο γενικός σχεδιασμός της επεξεργασίας δεδομένων (data path) παρουσιάζεται στην παρακάτω Εικόνα 1.2.1 :

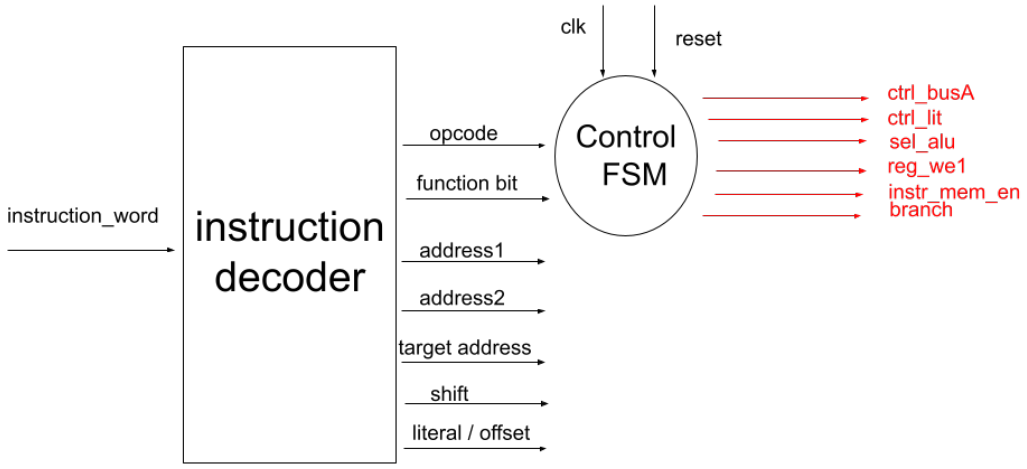


Εικόνα 1.2.1 - Ο κύριος πυρήνας του επεξεργαστή Robin

Ο κύριος δίαυλος δεδομένων, το busA, είναι είσοδος στο register file, ενώ το busB και το busC εξάγονται από το register file και εισάγονται στην ALU. Η άμεση διευθυνσιοδότηση παρέχεται μέσω ενός πολυπλέκτη δύο προς ένα (2:1), που μπορεί να επιλέξει μια αριθμητική τιμή 8bit (literal) ως όρισμα εισόδου στη μονάδα ALU. Το register file αποτελείται από 32 καταχωρητές των 8 bit, οι οποίοι είναι διαχωρισμένοι σε υποσύνολα καταχωρητών συστήματος, γενικών καταχωρητών και περιφερειακών καταχωρητών.

Το αρχείο καταχωρητών έχει σχεδιαστεί ως RAM πολλαπλών θυρών, με δύο θύρες διευθύνσεων για ανάγνωση (διεύθυνση1 και διεύθυνση2) και μία θύρα διεύθυνσης για εγγραφή (διεύθυνση 3). Με αυτόν τον τρόπο, διαβάζονται ταυτόχρονα δύο καταχωρητές (reg1, reg2) από το αρχείο καταχωρητών και το περιεχόμενό τους

εξάγεται στο busB και στο busC. Το σύστημα μπορεί να εγγράψει το περιεχόμενο του διαύλου δεδομένων (busA) στη διεύθυνση προορισμού(target address).



Εικόνα 1.2.2 - Αποκωδικοποιητής εντολών και η μονάδα ελέγχου (Control FSM)

Η διεύθυνση προορισμού μπορεί να είναι ή η διεύθυνση1 ή ο καταχωρητής W (καταχωρητής εργασίας) του συστήματος.

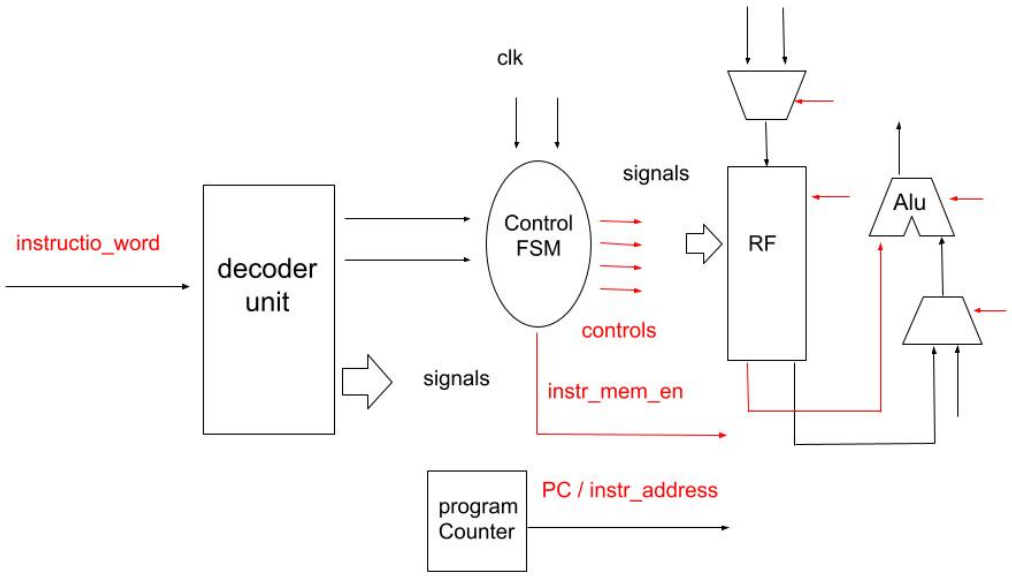
Ο αποκωδικοποιητής λαμβάνει τη λέξη της εντολής από τη μνήμη του προγράμματος και αποκωδικοποιεί το opcode και τους τελεστές της εντολής. Σε κάθε κύκλο εντολής τροφοδοτείται μια νέα εντολή στο κύκλωμα του αποκωδικοποιητή και χωρίζεται στα κατάλληλα πεδία: opcode, register address 1,register address 2 (ή literal value) και το function bit.

Το opcode και το function bit (fb) εισάγονται στην κύρια μονάδα ελέγχου FSM, η οποία αποκωδικοποιεί τα σήματα ελέγχου σε κάθε βήμα εκτέλεσης, ανάλογα με τον opcode και το fb.

Τα κύρια σήματα ελέγχου είναι τα σήματα επιλογής mux και alu, το σήμα ενεργοποίησης της εγγραφής καθώς και το σήμα ενεργοποίησης της μνήμης εντολών (βλέπε Εικόνα1.2.2). Η μονάδα ελέγχου έχει σχεδιαστεί ως μηχανή πεπερασμένων καταστάσεων (FSM) τεσσάρων κύκλων ρολογιού, σύμφωνα με τη βασική αρχή ότι όλες οι εντολές εκτελούνται στις ίδιες τέσσερις περιόδους του κύριου ρολογιού. Οι

τέσσερις φάσεις είναι οι τυπικές καταστάσεις λήψης-αποκωδικοποίησης-εκτέλεσης-αποθήκευσης (fetch-decode-execute-store). Η κάθε μία κατάσταση διαρκεί μία περίοδο ρολογιού.

Ο πυρήνας του επεξεργαστή Robin με τα κύρια μπλοκ του φαίνεται στο διάγραμμα της Εικόνας 1.2.3.



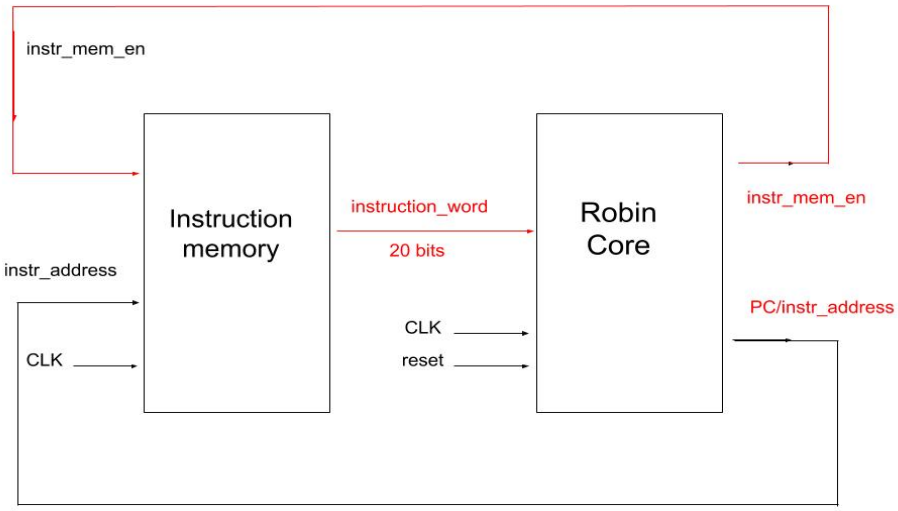
Εικόνα 1.2.3 - Ο πυρήνας του επεξεργαστή Robin με τα κύρια μπλοκ του. Ο μετρητής προγράμματος εφαρμόζεται ως ξεχωριστό κύκλωμα.

Ο πυρήνας του επεξεργαστή Robin αποτελείται από τη μονάδα αποκωδικοποιητή εντολών, τη μονάδα παραγωγής των σημάτων ελέγχου (control FSM), το αρχείο Καταχωρητών και τη μονάδα ALU. Στο ίδιο διάγραμμα, εμφανίζεται επίσης το κύκλωμα μετρητή προγράμματος.

Η επόμενη διεύθυνση εντολών αποκωδικοποιείται στη φάση αποκωδικοποίησης της εκτέλεσης της τρέχουσας εντολής. Για το σκοπό αυτό, ένας μετρητής προγράμματος αυξάνεται κατά την περίοδο ανάκτησης της τρέχουσας εντολής. Η τιμή του μετρητή προγράμματος χρησιμοποιείται για την παραγωγή της διεύθυνσης εντολής που είναι είσοδος στη μονάδα της μνήμης εντολών. Το σήμα ελέγχου που ενεργοποιεί τη μνήμη καθορίζεται στη φάση αποθήκευσης της τρέχουσας εντολής, ξεκινώντας την ανάγνωση της επόμενης εντολής.

Το κύκλωμα μετρητή προγράμματος εφαρμόζει επιπλέον λογική για την εκτέλεση εντολών διακλάδωσης και άλματος. Η μονάδα ελέγχου ξεκινά με μια κατάσταση αδράνειας (idle), όπου όλα τα σήματα έχουν λαμβάνουν τις αρχικές τους τιμές.

Το σύστημα οδηγείται ακόμη σε κατάσταση αδράνειας όταν ολοκληρωθεί η εκτέλεση του προγράμματος με εντολή τερματισμού.

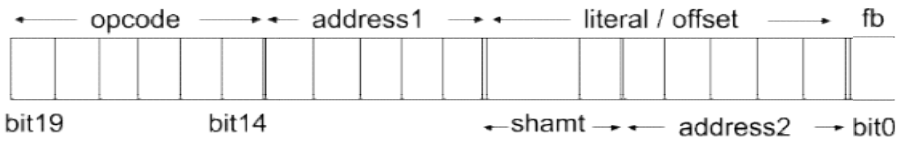


Εικόνα 1.2.4 - Οντότητα ανώτερου επιπέδου του επεξεργαστή , με μνήμη εντολών και κύκλωμα πυρήνα

Σε αυτή την έκδοση του μικροελεγκτή (v0), ο επεξεργαστής ακολουθεί μια αυστηρή αρχιτεκτονική Harvard, με μια ξεχωριστή Μνήμη Εντολών (Η αρχιτεκτονική Harvard είναι μια αρχιτεκτονική υπολογιστών με ξεχωριστούς διαδρόμους για εντολές και δεδομένα). Η βασική μνήμη εντολών έχει χωρητικότητα 1024 λέξεων εντολών, ενώ κάθε λέξη έχει πλάτος 20 bit. Επομένως, η διεύθυνση εντολής είναι 10 bit.

1.3 Instruction word

Η λέξη της δυαδικής εντολής αποτελείται από τα πεδία που εμφανίζονται στην Εικόνα 1.3.1. Διαφορετικοί τύποι εντολών χρησιμοποιούν διαφορετικά πεδία. Το κύριο σχήμα, ωστόσο, ξεκινά με το opcode, ακολουθούμενο από τη διεύθυνση 5-bit του πρώτου καταχωρητή και τελειώνει με μια διεύθυνση 5-bit του δεύτερου καταχωρητή, σύμφωνα με τη φόρμα: opcode <address1>, <address2>.



Εικόνα 1.3.1 – Πεδία του Instruction word

Ένα πεδίο 3-bit μεταξύ των δύο πεδίων διεύθυνσης ονομάζεται "shamt" και χρησιμοποιείται κυρίως για να φιλοξενήσει την παράμετρο shift στις εντολές ολίσθησης. Συνδυάζεται, επίσης, με το πεδίο address2, προκειμένου να αντιπροσωπεύει μια άμεση αριθμητική τιμή (literal), εύρους 8-bit, σε εντολές άμεσης διευθυνσιοδότησης.

Μια literal ή άμεση (immediate) τιμή αντιπροσωπεύει έναν αριθμό αντί για μια διεύθυνση. Υπό αυτήν την έννοια, η εντολή μετακινεί την κυριολεκτική (literal) τιμή 0x28 στον καταχωρητή με διεύθυνση 0x0A.

Η εντολή: movi <address1>, literal (e.g. movi 0x0A, 0x28)

Στην περίπτωση αυτή, ο assembler τοποθετεί τη διεύθυνση 0x0A στο πεδίο διεύθυνσης 1 και την άμεση τιμή 0x28 στον συνδυασμό 8-bit shamt & address2. Ο τελεστής «&» σημαίνει συνένωση.

Το παραπάνω πεδίο (shamt & address2) μπορεί να φιλοξενήσει επίσης μια μετατόπιση (offset) 8-bit για μια λειτουργία διακλάδωσης. Η κυριολεκτική τιμή στο πεδίο shamt&address2 δύναται να φιλοξενήσει μια μετατόπιση 8-bit για μια λειτουργία κλάδου. Η εντολή: shift <address1>, <address2>, shift_parameter

Η παράμετρος μετατόπισης μετατοπίζει το περιεχόμενο του καταχωρητή με τη διεύθυνση 2 από την παράμετρο μετατόπισης και τοποθετεί το αποτέλεσμα στη διεύθυνση 1. Ο assembler τοποθετεί την παράμετρο μετατόπισης στο shamt πεδίο. Υπό αυτήν την έννοια, η παράμετρος μετατόπισης μπορεί να κυμαίνεται από 0 έως 7. Το τελικό bit στη λέξη δυαδικής εντολής είναι το function bit ή fb. Αυτό το bit ορίζει έναν από τους εναλλακτικούς τρόπους εκτέλεσης μιας εντολής .

Συνολικά οι 64 εντολές μπορούν να εφαρμοστούν από το παραπάνω δυαδικό σχήμα εντολών. Στο αρχείο μητρώου (register file) μπορεί να είναι μόνο 32 καταχωρητές, λόγω των 5 bit στα πεδία διευθύνσεων.

Τέλος, το πεδίο διεύθυνσης 1 σε συνδυασμό με το πεδίο shamt και το πεδίο διεύθυνσης 2 μπορεί να φιλοξενήσει μια απόλυτη διεύθυνση (μέγιστο 13 bits) για μια λειτουργία άλματος.

Συμπερασματικά, διαφορετικοί τύποι εντολών κάνουν διαφορετική χρήση των πεδίων εντολών. Ανάλογα με την έννοια των διαφόρων πεδίων, αναγνωρίζουμε τρεις βασικούς τύπους οδηγιών:

1) opcode-reg-reg 2) opcode-reg-literal 3) opcode-address

1.4 Register file

Παρουσίαση του αρχείου καταχωρητών. Το αρχείο καταχωρητών (RF) αποτελείται από 32 καταχωρητές, οι οποίοι είναι προσβάσιμοι από πεδία 5 bit στη λέξη της δυαδικής εντολής. Τα πεδία αυτά τοποθετούνται μετά τον κωδικό εντολής (opcode).

Πίνακας 1–Ο Βασικός Πίνακας Καταχωρητών

Address	Register name	Description	Comments
00000	W reg	Working register	System register
00001	Status reg	zero, carry, gt, lt eq	System register
00010	IC	Interrupt Flags	System register
00011	PCL	Program counter low	System register
00100	PCH	Program counter high	System register
00101	Base address reg	Keeps RAM base addres	System register
00110	reg1	general register	general user register
00111	reg2	"	"
01000	reg3	"	"
01001	reg4	"	"
01010	reg5	"	"
01011	reg6	"	"
01100	reg7	"	"
01101	reg8	"	"
01110	reg9	"	"
01111	reg10	"	"
10000	reg11	"	"
10001	reg12	"	"
10010	reg13	"	"
10011	reg14	"	"
10100	reg15	"	"
10101	reg16	"	"
10110	preg1	Can be assigned to a peripheral	peripheral register
10111	preg2	"	"
11000	preg3	"	"
11001	preg4	"	"
11010	preg5	"	"
11011	preg6	"	"
11100	preg7	"	"
11101	preg8	"	"
11110	preg9	"	"
11111	preg10	"	"

Οι καταχωρητές χωρίζονται στις ακόλουθες 3 ομάδες, ανάλογα με το ρόλο τους, όπως παρουσιάζονται με χρωματικό κώδικα στον Πίνακα 1.

System Registers: Οι Καταχωρητές Συστήματος συνολικά είναι έξι:

1 . w → καταχωρητής εργασίας
2. Statusreg → καταχωρητές κατάστασης
3. InterruptFlags → σημαίες διακοπής
4. Program counter low → χαμηλός μετρητής προγράμματος
5. Program counter high → υψηλός μετρητής προγράμματος
6. Base address reg → Καταχωρητής βασικών διευθύνσεων

General Registers: Οι Γενικοί Καταχωρητές είναι συνολικά δεκαέξι, από το reg1 έως τον reg16. Οι συγκεκριμένοι καταχωρητές χρησιμοποιούνται είτε ως πηγή είτε ως προορισμός (source/target) .

Peripheral Registers: Τα περιφερειακά επέκτασης μπορούν να ομαδοποιηθούν σε δύο ή τέσσερις καταχωρητές ανά περιφερειακό. Είναι δέκα καταχωρητές. Μπορούν να χρησιμοποιηθούν ως κανονικοί γενικοί καταχωρητές, εάν δεν έχουν αντιστοιχία σε περιφερειακά κάποιων χρηστών.

1.5 Περιγραφή των καταχωρητών του συστήματος

Μεταξύ των καταχωρητών του συστήματος, σημαντικό ρόλο διαδραματίζουν τα ακόλουθα:

W register: Ο καταχωρητής W μπορεί να χρησιμοποιηθεί ως γενικός συσσωρευτής για τη μεταφορά αποτελεσμάτων alu μεταξύ των καταχωρητών. Η χρήση του συνδυάζεται με το bit λειτουργίας της λέξης εντολής. Εάν το bit λειτουργίας (fb) έχει οριστεί σε 1, τότε το αποτέλεσμα alu θα αποθηκευτεί στον καταχωρητή W. Εάν fb = 1, τότε το αποτέλεσμα θα αποθηκευτεί στον καταχωρητή reg1 (πεδίο 1 στη λέξη εντολής).

Base address register: Ο καταχωρητής βασικών διευθύνσεων μπορεί να χρησιμοποιηθεί για τη διατήρηση μιας αρχικής διεύθυνσης ενός πίνακα στη μνήμη και συνδυάζεται με μια μετατόπιση στο reg2 της εντολής ld ή st.

Σε μια έκδοση, μπορεί να διατηρήσει μια διεύθυνση εκκίνησης και να προσθέσει το offset (base + offset) . Σε μια άλλη έκδοση, μπορεί να κρατήσει τα 8 MSB μιας διεύθυνσης και να τα συνενώσει με το offset στο reg2 μιας εντολής ld ή st (επεκτείνει τη μνήμη RAM έως 64K)

STATUS register: Ο καταχωρητής status είναι ένας καταχωρητής συστήματος που αποθηκεύει τις βασικές σημαίες, οι οποίες προέρχονται από την μη καταχωρημένη αριθμητική λειτουργία της alu. Τα status bits περιγράφονται παρακάτω:

Πίνακας 2 - Status bit

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LS	shift	gt	eq	It	OV	Zero(Z)	Carry (C)

Carry Flag: Η σημαία κρατουμένου (C) γίνεται "1" όταν παράγεται bit κρατουμένου / δανεικού στο νιοστό bit μιας μη προσημασμένης πρόσθεσης ή αφαίρεσης.

Zero Flag: Η μηδενική σημαία (Z) ορίζεται όταν το αποτέλεσμα alu είναι μηδέν.

Over flow: Το bit υπερχείλισης (OV) υποδηλώνει υπερχείλιση όταν συμβαίνει και θα πρέπει να λαμβάνεται υπόψη σε προσημασμένους αριθμούς.

gt_eq it: Τα bits σύγκρισης gt, eq, lt ρυθμίζονται μόνο μετά από μια λειτουργία sub ή cmp. Στην πραγματικότητα, το "cmp reg1, reg2" εκτελεί μια αφαίρεση reg1-reg2 και ορίζει τα bit σύγκρισης βάσει στο αποτέλεσμα carry και zero bit. Πιο συγκεκριμένα : gt = carry XOR zero, eq = zero, lt = NOT(carry)

Shift: Το Shift bit (bit5) φιλοξενεί το μετατοπισμένο MSB ενός καταχωρητή χρηστών (user register) μετά την εκτέλεση μιας εντολής shift-left (sl). Φιλοξενεί το μετατοπισμένο LSB ενός καταχωρητή χρηστών (user register) μετά την εκτέλεση μιας εντολής shift-right (sr).

LS: Το Literal / Offset Sign bit (LS) φιλοξενεί το πιο σημαντικό bit της literal/offset τιμής. Ανάλογα με την τιμή του και την τιμή του παραγόμενου bit μεταφοράς κατά την εκτέλεση μιας εντολής κλάδου, το πεδίο pc_high αυξάνεται, μειώνεται ή παραμένει αμετάβλητο.

Ο καταχωρητής STATUS παραμένει αμετάβλητος έως ότου μια νέα λειτουργία της alu επηρεάσει τα bit της.

1.6 Σύνολο Εντολών

Το σύνολο των εντολών που χρησιμοποιεί ο επεξεργαστής Robin, βρίσκεται στον παρακάτω πίνακα:

Πίνακας 3 - Σύνολο Εντολών του V-RISC

Category	Instruction name	Assembly code	opcode	fb	Assembly syntax	Operation	Comments
Data transfer (register-register)	move immediate	movi	000001	0	movi reg1, 0x28	w/reg1 <- literal	Move literal value to reg1 (immediate)
	move	mov	000010	0	mov reg1, reg2	w/reg1 <- reg2	Move content of reg2 to reg1 (register direct)
	move indirect	mind	000000	0	mind reg1, *reg2	w/reg1 <- [reg2]	Move the content of register address in base reg into reg1 (register indirect)
Data transfer (register-memory)	load	ld	000011	0/1[a]	ld reg1, *reg2	reg1<-[reg2]	Load reg1 with the content of address in reg2 (memory indirect - ram version)
	store	st	000100	0/1[a]	st reg1, &reg2	reg1->[reg2]	Store reg1 to the address in reg2 - ram version
Arithmetic	add	add	000101	0/1[b]	add reg1, reg2	w/reg1 <- reg1+reg2	Affects zero, ov and carry bits
	sub	sub	000110	0/1[b]	sub reg1, reg2	w/reg1<-reg1-reg2	Affects gt/eq/lt, zero, ov and carry bits
	add with carry c	addc	000111	0	addc reg1, reg2	w/reg1 <- reg1 + reg2 + carry	Affects zero, ov and carry bits
	subtract with borrow	subc	001000	0	Subc reg1, reg2	w/reg1 <- reg1 - reg2 - carry	Affects gt/eq/lt, zero, ov and carry bits
	add immediate	addi	001001	0/1[b]	addi reg1, 0x28	w/reg1<-reg2+literal	
	sub immediate	subi	001010	0/1[b]	sub reg1, 0x28	w/reg1<-reg1-literal	

Logical	and	and	001011	0/1[b]	and reg1, reg2	w/reg1<- reg1 and reg2	Bitwise and
	or	or	001100	0/1[b]	or reg1, reg2	w/reg1<- reg1 or reg2	Bitwise or
	and immediate	andi	001101	0/1[b]	andi reg1, 0x28	w/reg1 <- reg1 and literal	
	or immediate	ori	001110	0/1[b]	ori reg1, 0x28	w/reg1 <- reg1 or literal	
	Shift left	sl	001111	0	sl reg1, reg2, 2	w/reg1<-reg2 << shift (0-7). status(6)<-reg2(7)	shifted MSB in shift-bit b5 of status reg, i.e. status(5)<-reg2(7)
	Shift right	sr	010000	0	sr reg1, reg2, 1	w/reg1<-reg2 >> shift(0-7). status(6)<-reg2(0)	Shifted MSB in b5 of status reg. i.e. status(5)<-reg2(0)
	compare	cmp	010001	1	cmp reg1, reg2	Set gt/lt/eq bits of STATUS reg (W <- reg1-reg2)	Compare register contents and set status bits. Affects W.
Conditional and unconditional branch	Branch on equal	beq	010010	1	beq offset	If reg1=reg2 in previous cmp command, then pc<- pc+offset (±127)	Branch type 1: PC-relative, +/- 127 commands
	Branch on greater than	bgt	010011	1	bgt offset	If *reg1>*reg2 in previous comp then pc<-pc+offset (±127)	Branch type 1: PC-relative, +/- 127 commands
	jump	jmp	010100	0/1[c]	jmp address	jump to address	Branch type 2: absolute 10-bit program address
	Call subroutine	call	010101	0	call address	Push PC in stack and jump to subroutine	Absolute addressing 10-bit
	Return from subroutine	ret	010110	0	ret	Pop PC from stack and return from subroutine	
other	Clear register	clr	010111	0	clr reg	Clear all bits of register	Only in expanded set
	No operation	nop	011000	0	nop	Do nothing	Only in expanded set
	End execution	end	111111	0	end	Go to the idle state and sleep	

[1] Για Function bit =1

Χρησιμοποιούν τον καταχωρητή base-address ως διεύθυνση βάσης και περιεχόμενο reg1 ως offset. Η μνήμη επεκτείνεται στα 64K σε έκδοση base&offset. Επεκτείνεται στα 512 bytes σε έκδοση base + offset. Με Function bit = 0, είναι προσβάσιμα μόνο τα 256 byte από τη μνήμη.

[2] Για Function bit = 1

Το αποτέλεσμα αποθηκεύεται στον καταχωρητή εργασίας w.

[3] Για Function bit = 0

Η branch address βρίσκεται στην πρώτη σελίδα της μνήμης προγράμματος (0-8191). Με τη συνάρτηση bit = 1 η branch address βρίσκεται στη δεύτερη σελίδα της μνήμης προγράμματος (8192-16383).

Κεφάλαιο 2

Προδιαγραφές του συμβολομεταφραστή

Στο δεύτερο κεφάλαιο γίνεται αναφορά στο λογισμικό καθώς και στο περιβάλλον μέσα στο οποίο δημιουργήθηκε η παρούσα πτυχιακή εργασία. Επίσης, γίνεται παρουσίαση με παραδείγματα των διάφορων τύπων που μπορεί να έχει μια εντολή, με βάση τους οποίους σχεδιάστηκε ο συμβολομεταφραστής. Στη συνέχεια, αναλύονται τα τμήματα που περιέχει μια εντολή, με βάση τα οποία γίνεται η αντιστοίχιση ανάμεσα στην assembly και τη γλώσσα μηχανής.

2.1 Γλώσσα προγραμματισμού ανάπτυξης κώδικα

Η παρούσα πτυχιακή εργασία αναπτύχθηκε με την γλώσσα προγραμματισμού Python και χρησιμοποιήθηκε η 3η έκδοση της. Η Python είναι μια γλώσσα με μεγάλη χρηστική αξία στις μέρες μας. Εφαρμόζεται σε πολλά πεδία,επιχειρήσεις και projects.

Κάποια από τα πεδία αυτά είναι εφαρμογές Web Development, Data Analytics, Machine Learning και πολλές άλλες. Πολύ γνωστές web εφαρμογές όπως η Google, το Instagram και το Youtube χρησιμοποιούν την Python για δικούς τους σκοπούς. Σαν γλώσσα προγραμματισμού δεν είναι πολύπλοκη στην ανάγνωση και στην γραφή της. Έχει μεγάλη κοινότητα χρηστών με αποτέλεσμα στο διαδίκτυο να υπάρχει πολύ βοηθητικό υλικό μέσω του οποίου έχει δημιουργηθεί και ένα ικανοποιητικά μεγάλο δίκτυο υποστήριξης. Έτσι, ο κάθε προγραμματιστής οποιουδήποτε επιπέδου (αρχάριος ή προχωρημένος) μπορεί εύκολα να βρει λύση, στην περίπτωση που τυχόν συναντήσει δυσκολία κατά την διαδικασία του προγραμματισμού. Τέλος, έχει εκτενείς βιβλιοθήκες (libraries), που κάνουν το προγραμματισμό πιο εύκολο.

2.2 Περιβάλλον ανάπτυξης κώδικα

Το περιβάλλον της PyCharm είναι η εφαρμογή που χρησιμοποιήθηκε για να γραφούν και να τρέξουν οι εφαρμογές της Python. Η PyCharm είναι ένα ολοκληρωμένο περιβάλλον για την ανάπτυξη (IDE) που χρησιμοποιείται στον προγραμματισμό, ειδικά για την γλώσσα Python. Επίσης, παρέχει ανάλυση κώδικα, ένα γραφικό πρόγραμμα εντοπισμού σφαλμάτων και έναν ενσωματωμένο ελεγκτή unit. Ένα προτέρημα της PyCharm είναι πως μπορεί να λειτουργήσει σε διάφορες πλατφόρμες , όπως τα Windows, Linux και macOS.

2.3 Τύποι εντολών

Αρχικά, για το συγκεκριμένο πρόγραμμα θα αναλύσουμε τα βασικά χαρακτηριστικά, στα οποία βασίστηκε και δομήθηκε ο συγκεκριμένος συμβολομεταφραστής. Υπάρχουν τρεις τύποι για την δομή-σύνταξη των εντολών:

1) opcode-reg-reg 2) opcode-reg-literal 3) opcode-address

Στην πρώτη περίπτωση, είναι η πρώτη εντολή το opcode από τον Πίνακα 2 και μετά ακολουθούν 2 registers, ο ένας μετά τον άλλο από τον Πίνακας 1 που τους διαχωρίζει ένα κόμμα. Για παράδειγμα : mov reg1, reg2

Στην δεύτερη περίπτωση έχουμε την πρώτη εντολή το opcode από τον Πίνακα 2. Στη συνέχεια ακολουθεί ένας register από τον Πίνακα 1 και μετά κάποιο literal. Όπου literal είναι ένας δεκαεξαδικός αριθμός δύο ψηφίων. Ο register και το literal διαχωρίζονται κι αυτά με κόμμα . Για παράδειγμα : movi reg1, 0xAF

Στην τρίτη περίπτωση έχουμε την πρώτη εντολή το opcode από τον Πίνακα 2 και στη συνέχεια ακολουθεί μια διεύθυνση η οποία είναι σε δυαδική μορφή. Για παράδειγμα : jmp b0000110001

2.4 Mnemonic – Opcode

Το opcode είναι η δυαδική μορφή της εντολής assembly ή αλλιώς mnemonic. Ο χρήστης είναι αυτός, ο οποίος πληκτρολογεί την εντολή assembly. Όλες οι εντολές που μπορούν να χρησιμοποιηθούν σαν mnemonic βρίσκονται στην πρώτη στήλη του Πίνακα 4 και είναι οι παρακάτω:

Πίνακας 4 - Αντιστοιχία Κώδικα Assembly με το Opcode

Assembly code	Opcode
movi	OOOOO1
mov	OOOO1O
mind	OOOOOO
ld	OOOO11
st	OOO1OO
add	OOO1O1
sub	OOO11O
addc	OOO111
subc	OO1OOO
addi	OO1OO1
subi	OO1O1O
and	OO1O11
or	OO11OO

Assembly code	Opcode
andi	OO11O1
ori	OO111O
sl	OO1111
sr	O1OOOO
cmp	O1OOO1
beq	O1OO1O
bgt	O1OO11
jmp	O1O1OO
call	O1O1O1
ret	O1O11O
clr	O1O111
nop	O11OOO
end	111111

Υπάρχουν εντολές που αποτελούνται μόνο από την εντολή assembly, δηλαδή το mnemonic, αυτές είναι οι : «ret», «nop», «end».

2.5 Είδη ορισμάτων που μπορεί να πάρει το πρώτο operand

Στο σημείο αυτό, θα αναφερθούν οι διάφορες κατηγορίες εντολών που μπορούμε να χρησιμοποιήσουμε σαν πρώτο operand, το οποίο είναι το δεύτερο όρισμα κάθε εντολής. Για να συμβεί αυτό βασιζόμαστε στην προηγούμενη εντολή assembly, το mnemonic, έτσι ώστε να οριστεί η κατηγορία του. Οι διάφοροι τύποι που μπορεί να πάρει παρατίθενται παρακάτω και είναι οι εξής:

1.Register name

Η πρώτη κατηγορία πεδίου είναι το όνομα του καταχωρητή (Register Name). Βασιζόμενοι στον Πίνακα 1, οι εντολές που μπορούν να χρησιμοποιηθούν σαν πρώτο operand βρίσκονται στην πρώτη στήλη.

Πίνακας 5 - Αντιστοιχία Register name - Address

Register name	Address
W reg	OOOOO
Status reg	OOOO1
IC	OOO1O
PCL	OOO11
PCH	OO1OO
Base address reg	OO1O1
reg1	OO11O
reg2	OO111
reg3	O1OOO
reg4	O1OO1
reg5	O1O1O
reg6	O1O11
reg7	O11OO
reg8	O11O1
reg9	O111O
reg10	O1111

Register name	Address
reg11	1OOOO
reg12	1OOO1
reg13	1OO1O
reg14	1OO11
reg15	1O1OO
reg16	1O1O1
preg1	1O11O
preg2	1O111
preg3	11OOO
preg4	11OO1
preg5	11O1O
preg6	11O11
preg7	111OO
preg8	111O1
preg9	1111O
preg10	11111

Η αναπαράσταση των διευθύνσεων του Πίνακα 5 μπορεί να γίνει μέσω τριών διαφορετικών αριθμητικών τύπων που αναλύονται παρακάτω.

2. Δεκαεξαδικός αριθμός

Στη δεύτερη κατηγορία, ως πρώτος τελεστέος(operand) μπορεί να γίνει ένας δεκαεξαδικός αριθμός. Ο αριθμός αυτός έχει ένα όριο της τάξης των δύο ψηφίων. Το εύρος των αριθμών, με βάση τον περιορισμό των ψηφίων, μπορεί να ξεκινά από το '00' και να φτάνει έως τον'1F' εφόσον ο αριθμός των καταχωρητών είναι 31. Το αναγνωριστικό πρόθεμα για κάθε δεκαεξαδικό αριθμό είναι το «0x» και στην συνέχεια ακολουθεί ο αριθμός. Για παράδειγμα: 0x1B

3. Δυαδικός αριθμός

Στην τρίτη κατηγορία, ως πρώτος τελεστέος μπορεί να βρίσκεται ένας δυαδικός αριθμός. Το όριο των δυαδικών ψηφίων που μπορεί να έχει είναι τα πέντε ψηφία. Το εύρος των αριθμών, με βάση τον περιορισμό των πέντε ψηφίων, μπορεί να ξεκινά από το '00000' και το ανώτατο όριο του να είναι το '11111'. Τέλος, το αναγνωριστικό πρόθεμα για τους δυαδικούς αριθμούς είναι το «b», το οποίο είναι ο αρχικός χαρακτήρας από τη λέξη binary. Για παράδειγμα : b01010

4. Δεκαδικός αριθμός

Στην τέταρτη και τελευταία κατηγορία αριθμών του πρώτου τελεστέου, μπορεί να βρίσκεται ένας δεκαδικός αριθμός. Οι δεκαδικοί αριθμοί έχουν όριο τα δύο ψηφία. Το εύρος των αριθμών μπορεί να είναι από το 0 έως το 31. Το αναγνωριστικό πρόθεμα για τους δεκαδικούς αριθμούς είναι το «d», από τον πρώτο χαρακτήρα της λέξης decimal. Για παράδειγμα : d12

Στον παρακάτω Πίνακα 6 παρουσιάζονται συγκεντρωμένα τα προθέματα για κάθε κατηγορία αριθμών.

Πίνακας 6 - Προθέματα για κάθε τύπο συστήματος αρίθμησης

Δεκαεξαδικός	0x
Δυαδικός	b
Δεκαδικός	d

2.6 Κατηγορίες εντολών που απαιτούν συγκεκριμένους τύπους πρώτου operand

Οι κατηγορίες των διαφόρων εντολών, που μπορούν να πάρουν συγκεκριμένους από τους παραπάνω τύπους ως ορίσματα, παρατίθενται παρακάτω.

1. Εντολές Immediate

Η πρώτη κατηγορία που παρουσιάζεται είναι οι εντολές Immediate. Οι συγκεκριμένες εντολές έχουν ως κοινό χαρακτηριστικό τον τελευταίο χαρακτήρα τους, ο οποίος είναι ο χαρακτήρας «i», από τον πρώτο χαρακτήρα της λέξης immediate. Οι εντολές αυτές μεταφέρουν το literal (το δεύτερο operand) στο πρώτο operand άμεσα. Το σύνολο των εντολών που απαρτίζουν την συγκεκριμένη κατηγορία είναι: «movi» , «addi» , «subi» , «andi» , «ori»

Το πρώτο operand στην κατηγορία των εντολών Immediate μπορεί να πάρει τιμές από τον Πίνακα 5 του Register File. Επίσης, μπορεί να πάρει κάποιο δυαδικό αριθμό των οκτώ bit, όπως και κάποιο διψήφιο δεκαεξαδικό αριθμό.

2. Εντολές Offset

Στην δεύτερη κατηγορία ανήκουν οι εντολές που έχουν ως όρισμα του πρώτου operand κάποιο offset. Οι εντολές που ανήκουν στη δεύτερη περίπτωση είναι δύο: η «beq» και η «bgt».

Το offset μπορεί να είναι μόνο όρισμα δεκαδικού αριθμού. Στην συγκεκριμένη περίπτωση οι δεκαδικοί αριθμοί είναι προσημασμένοι. Το εύρος των αριθμών που μπορούν να πάρουν οι εντολές με όρισμα offset, είναι από το -128 έως το +127. Για να γίνει ένας διαχωρισμός σε θετικούς και αρνητικούς αριθμούς, θεωρούμε το 0 ως θετικό αριθμό και ως σημείο αναφοράς. Οι θετικοί αριθμοί απαρτίζονται από το 0 έως και το +127, ενώ οι αρνητικοί απαρτίζονται από το -1 έως και το -128. Για να διαχωριστούν σωστά και κατά την συγγραφή κώδικα, ορίζουμε ένα πρόθεμα για κάθε κατηγορία. Οι θετικοί αριθμοί θα έχουν ως πρόθεμα το «d+», ενώ το πρόθεμα για τους αρνητικούς θα είναι «d-».

3.Address

Η τρίτη κατηγορία είναι αυτή που ως πρώτο operand μπορεί να πάρει όρισμα με τύπο address. Η εντολές αυτές είναι δύο η «jmp» και η «call». Η συγκεκριμένη έχει τη δυνατότητα να πάρει τιμές σε δυαδική μορφή των δέκα bit. Έτσι, μπορεί να φορτώσει τον απαριθμητή προγράμματος, που έχει εύρος 10-bit.

4. Όλοι οι τύποι πρώτου operand

Στην τελευταία κατηγορία συμπεριλαμβάνονται όλες οι εντολές, οι οποίες δεν έχουν κάποιο αριθμητικό περιορισμό για τον τύπο περιεχομένου του ορίσματος που μπορεί να πάρουν ως πρώτο operand. Οι εντολές αυτές λαμβάνουν ως πρώτο όρισμα κάποιον καταχωρητή reg.

Οι εντολές αυτές είναι οι παρακάτω: «mov», «mind», «ld», «st», «add», «sub», «addc», «subc», «addi», «subi», «and», «or», «andi», «ori», «sl», «sr», «cmp».

Υπάρχουν κάποιες εντολές αποτελούμενες μόνο από την εντολή assembly και το πρώτο operand. Οι εντολές αυτές δεν έχουν συνέχεια, δηλαδή δεν συμπεριλαμβάνουν δεύτερο operand. Αυτές είναι οι εξής: «beq», «bgt», «jmp», «clr».

2.7 Είδη ορισμάτων που μπορεί να πάρει το δεύτερο operand

Όπως το πρώτο, έτσι και το δεύτερο operand μπορεί να πάρει όλους τους τύπους ορισμάτων (κάποιο register name, ή κάποιο δυαδικό, δεκαδικό ή δεκαεξαδικό αριθμό). Οι εντολές που μπορούν να πάρουν όλου του είδους τα ορίσματα σαν δεύτερο operand είναι οι εξής: «mov», «mind», «ld», «st», «add», «sub», «addc», «subc», «and», «or», «sl», «sr», «cmp».

2.8 Κατηγορία εντολών που απαιτεί συγκεκριμένο τύπο δεύτερου operand

Στην περίπτωση αυτή, υπάρχει μόνο μία κατηγορία που απαιτεί συγκεκριμένο όρισμα ως δεύτερο operand. Η συγκεκριμένη κατηγορία εντολών είναι αυτή με τις εντολές Immediate. Όπως αναφέρθηκε παραπάνω, οι συγκεκριμένες εντολές ξεχωρίζουν διότι ο τελευταίος χαρακτήρας τους είναι το «i» και είναι οι παρακάτω πέντε: «movi», «addi», «subi», «andi», «ori».

Οι εντολές αυτές, μπορούν να πάρουν δύο συγκεκριμένους τύπους αριθμών, σε δυαδική, δεκαδική ή δεκαεξαδική μορφή. Το πρόθεμα είναι το ίδιο στην κάθε κατηγορία με το πρώτο operand. Για τους δυαδικούς αριθμούς είναι το «b», για τους δεκαδικούς το «d» και για τους δεκαεξαδικούς αριθμούς το «0x».

2.9 Εντολές που έχουν shamt

Το shamt είναι ένα επιπλέον πεδίο, το οποίο βρίσκεται αμέσως μετά από το δεύτερο operand. Το πεδίο shamt έχουν μόνον δύο εντολές, η «sl» και η «sr».

Το είδος των ορισμάτων, που έχει τη δυνατότητα να πάρει το shamt, είναι δεκαδικοί αριθμοί. Το εύρος των τιμών έχει οριοθετηθεί από το 0 έως τον αριθμό 7. Στο shamt, πριν το όρισμα του δεκαδικού αριθμού, τοποθετείται το πρόθεμα «d». Για παράδειγμα: sl reg1, reg2, d3

2.10 Function bit

Το function bit(fb) αποτελείται, όπως λέει και το όνομά του, από ένα bit. Όλες οι εντολές ανεξαρτήτως μεγέθους και τύπου(Πίνακας 2), έχουν function bit. Το function bit μπορεί να πάρει δυαδικές τιμές (0 ή 1), που αποτελούνται από ένα bit. Οι εντολές χωρίζονται σε τρεις κατηγορίες, ανάλογα με το χαρακτηριστικό function bit.

Η πρώτη κατηγορία εντολών έχει ως όρισμα fb το 0 και είναι οι παρακάτω: «mov», «movi», «mind», «sl», «sr», «call», «ret» , «clr», «nop», «end».

Η δεύτερη κατηγορία εντολών μπορεί να πάρει ως όρισμα 0 ή 1 και είναι οι: «ld», «st», «add», «sub», «addc», «subc», «addi», «subi», «and», «or», «andi», «ori», «jmp».

Η τρίτη και τελευταία κατηγορία εντολών, που το function bit μπορεί να πάρει ως όρισμα 1, είναι οι εξής τρείς: «cmp», «beq», «bgt».

Στις περιπτώσεις που το function bit είναι προκαθορισμένο, 0 ή 1 (σύμφωνα με τον Πίνακα 3), δεν το πληκτρολογεί ο χρήστης. Στην δεύτερη περίπτωση που το όρισμα μπορεί να είναι 0 ή 1, ο κώδικας το βλέπει σαν μία προκαθορισμένη τιμή 0. Τέλος, το function bit δεν έχει κάποιο πρόθεμα στην εντολή της assembly.

2.11 Σύνταξη των εντολών για τον assembler

Η συγγραφή των εντολών γίνεται σε ένα κειμενογράφο(Text Editor). Η σύνταξη της συγγραφής των εντολών είναι απλή για τον χρήστη. Αρχικά, γράφουμε την assembly εντολή, δηλαδή το mnemonic. Στη συνέχεια, ακολουθεί ένα κενό, το οποίο ξεχωρίζει την εντολή από το πρώτο operand. Έπειτα, γράφεται το πρώτο operand με τον ανάλογο τύπο ορίσματος, όπως αναφέρθηκε παραπάνω. Ακολουθεί ένα κόμμα και στη συνέχεια πάλι κενό, ώστε να ξεχωρίσει το πρώτο operand από το δεύτερο. Εν συνεχεία, γράφεται το δεύτερο operand, όπου σημαίνει και το τέλος των περισσότερων εντολών. Στην περίπτωση που η εντολή περιέχει shamt, τότε τοποθετείται ένα κόμμα αμέσως μετά το δεύτερο operand, στη συνέχεια αφήνουμε ένα κενό και τέλος γράφουμε το shamt, ως δεκαδικό. Όταν φτάσουμε στο τέλος όλων των εντολών, ο κέρσορας πρέπει να βρίσκεται στο τέλος του τελευταίου χαρακτήρα της τελευταίας εντολής. Αν βρίσκεται σε μια κενή γραμμή κάτω από τις εντολές, εμφανίζεται μήνυμα λάθους. Ακολουθούν παραδείγματα με όλα τα ενδεχόμενα εντολών:

1. end
2.clr reg1
3.mov reg3, reg4
4.sl reg1, reg2, d2

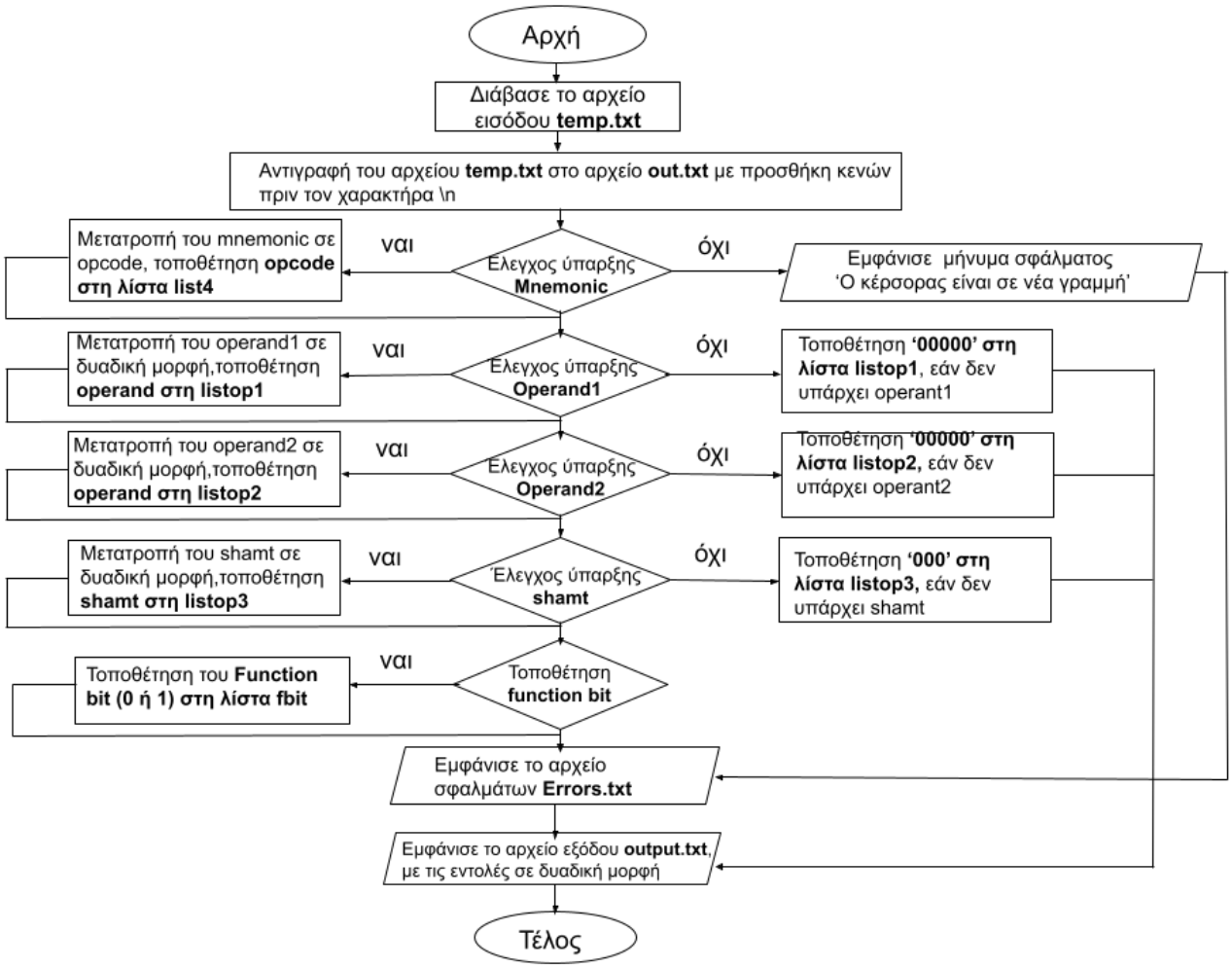
Κεφάλαιο 3

Τεχνικές ανάπτυξης κώδικα του συμβολομεταφραστή

Το τρίτο κεφάλαιο είναι αφιερωμένο στην δομή και τη λειτουργία του προγράμματος δημιουργίας του συμβολομεταφραστή καθώς και του γραφικού περιβάλλοντος. Αρχικά παρουσιάζεται ένα διάγραμμα ροής και μια σύντομη περιγραφή της λειτουργίας του. Ακολουθεί η αναλυτική περιγραφή του κώδικα για τη δημιουργία του συμβολομεταφραστή(assembler) καθώς και του γραφικού περιβάλλοντος του Text Editor, μέσα από το οποίο λειτουργεί ο Assembler. Τέλος, παρατίθενται εικόνες που παρουσιάζουν το γραφικό περιβάλλον με τον απαραίτητο σχολιασμό.

3.1 Διάγραμμα Ροής του προγράμματος συμβολομεταφραστή

Το παρακάτω διάγραμμα ροής περιγράφει συνοπτικά τη λειτουργία του συμβολομεταφραστή(assembler). Η έναρξη του διαγράμματος δηλώνεται μέσω του ελλειπτικού σχήματος που περιέχει τη λέξη 'Αρχή'.



Διάγραμμα ροής του προγράμματος συμβολομεταφραστή

Το πρόγραμμα ξεκινά με την εισαγωγή του αρχείου εισόδου, το οποίο ονομάζεται «temp.txt» και περιέχει τις εντολές σε γλώσσα assembly. Το αρχείο εισόδου αντιγράφεται σε ένα νέο αρχείο κειμένου με την ονομασία «out.txt», με μία τροποποίηση. Η μετατροπή που συμβαίνει στο νέο αρχείο αφορά το χαρακτήρα αλλαγής γραμμής το «\n», όπου πριν από αυτόν προσθέτονται μερικά κενά. Στη συνέχεια, τα περιεχόμενα του «out.txt» διαχωρίζονται σε λίστες. Η πρώτη κατηγορία των στοιχείων προς μετατροπή είναι των εντολών mnemonic.Το mnemonic είναι η

πρώτη εντολή που υπάρχει οπωσδήποτε στη δομή των εντολών ως πρώτο στοιχείο. Έτσι, το mnemonic μετατρέπεται στη δυαδική μορφή που ονομάζεται opcode και αποθηκεύεται στη λίστα «list4». Στην περίπτωση που ο κέρσορας έχει μετακινηθεί σε νέα γραμμή, η οποία δεν περιέχει κάποια εντολή, τότε εμφανίζεται μήνυμα λάθους. Η επομένη εντολή αφορά τον έλεγχο του πρώτου operand, εάν υπάρχει τότε μετατρέπεται σε δυαδική μορφή, η οποία αποθηκεύεται στη λίστα «listop1». Στην περίπτωση που δεν υπάρχει στη δομή της εντολής το στοιχείο αυτό, στην ίδια λίστα αποθηκεύονται πέντε μηδενικά «00000», που δηλώνουν την απουσία του στοιχείου από την εντολή. Για το δεύτερο operand ισχύει η ίδια διαδικασία με το πρώτο, εάν υπάρχει, μετατρέπεται σε δυαδική μορφή. Η λίστα αποθήκευσης του ονομάζεται «listop2». Εάν δεν υπάρχει, στην ίδια λίστα αποθηκεύονται πέντε μηδενικά «00000», που υποδεικνύουν τη έλλειψη του δεύτερου operand. Κάποιες από τις εντολές στη δομή τους περιέχουν και το στοιχείο shamt. Το συγκεκριμένο στοιχείο εάν υπάρχει μετατρέπεται σε δυαδική μορφή και το αποτέλεσμα της μετατροπής αποθηκεύεται στη λίστα «listop3». Σε αντίθετη περίπτωση στην ίδια λίστα τοποθετούνται τρία μηδενικά «000». Η τελευταία προσθήκη στη δομή των εντολών, είναι το στοιχείο function bit, το οποίο δεν χρειάζεται κάποια μετατροπή, καθώς είναι είτε 0 είτε 1 και τοποθετείται στη λίστα «fbit».

Η διαδικασία και τα κριτήρια μετατροπής του κάθε στοιχείου παραθέτονται στο κεφάλαιο 3.2.5 που ακολουθεί.

Κλείνοντας το διάγραμμα ροής δημιουργούνται δύο αρχεία εξόδου. Το πρώτο είναι το αρχείο σφαλμάτων, όπου περιέχονται τα σφάλματα του κώδικα και ονομάζεται «Errors.txt». Το δεύτερο είναι το αρχείο εξόδου, όπου σε αυτό αποθηκεύονται τα αποτελέσματα σε δυαδική μορφή από τις λίστες αποθήκευσης(list4, listop1, listop2, listop3, fbit). Το αρχείο αυτό, έχει την ονομασία «output.txt». Το διάγραμμα ροής κλείνει με το σχήμα της έλλειψης που περιέχει τη λέξη ‘Τέλος’ που ορίζει και το τέλος του προγράμματος.

3.2 Η δομή του προγράμματος για το συμβολομεταφραστή (assembler)

Στο κεφάλαιο αυτό γίνεται η παρουσίαση του συμβολομεταφραστή(assembler) και θα αναφερθούν όλα τα κύρια χαρακτηριστικά του καθώς και η δομή του. Ο συμβολομεταφραστής είναι το κυρίως πρόγραμμα, με το οποίο γίνεται η μετατροπή ενός αρχείου κειμένου, γραμμένο με κώδικα assembly σε δυαδική μορφή. Προκειμένου το συγκεκριμένο πρόγραμμα να γίνει κατανοητό είναι χωρισμένο σε τμήματα, όπου το καθένα από αυτά έχει μια συγκεκριμένη λειτουργία. Οι λειτουργίες αυτές αναφέρονται παρακάτω.

3.2.1 Βιβλιοθήκες

Οι βιβλιοθήκες της Python είναι μια συλλογή λειτουργιών και μεθόδων που επιτρέπουν στον κώδικα να εκτελεί πολλές ενέργειες με τη χρήση μιας δεσμευμένης λέξης. Αυτό έχει ως όφελος για τον προγραμματιστή τη γραφή λιγότερων γραμμών κώδικα. Στο πρόγραμμα για τον συμβολομεταφραστή γίνεται η εισαγωγή δύο βασικών βιβλιοθηκών που αφορούν τη λειτουργία του προγράμματος. Για να γίνει η δήλωση της εισαγωγής των βιβλιοθηκών χρησιμοποιείται αρχικά η λέξη «import», έπειτα, ακολουθεί ένα κενό και μετά το όνομα της εκάστοτε βιβλιοθήκης.

Η πρώτη βιβλιοθήκη έχει το όνομα «re», από τα αρχικά της λέξης regular. Η συγκεκριμένη βιβλιοθήκη δίνει τη δυνατότητα στον προγραμματιστή να χρησιμοποιεί regular εκφράσεις, δηλαδή προσφέρει ένα σύνολο λειτουργιών, το οποίο επιτρέπει να γίνεται η αναζήτηση κάποιας συμβολοσειράς.

Η δεύτερη βιβλιοθήκη έχει το όνομα «sys». Η ονομασία της προέρχεται από τα αρχικά της λέξης system. Η συγκεκριμένη παρέχει συναρτήσεις και μεταβλητές που χρησιμοποιούνται για τον χειρισμό διαφορετικών τμημάτων του περιβάλλοντος εκτέλεσης της Python.

3.2.2 Λεξικό

Η κύρια δυνατότητα που προσφέρει το λεξικό είναι η αποθήκευση τιμών σε ζεύγη. Οι τιμές και τα δεδομένα αποθηκεύονται σε κλειδιά και τιμές. Ένα λεξικό είναι μια συλλογή μη ταξινομημένη, η οποία μπορεί να αλλάξει οποιαδήποτε στιγμή, αλλά δεν μπορεί να περιέχει διπλότυπα. Με τον όρο διπλότυπα εννοούμε τις λέξεις κλειδιά, οι

οποίες πρέπει να είναι μοναδικές. Η δήλωση ενός λεξικού γίνεται γράφοντας πρώτα το όνομά που έχει επιλέξει ο εκάστοτε προγραμματιστής. Μια προγραμματιστική τεχνική είναι το όνομα του λεξικού να σχετίζεται με το περιεχόμενο του. Ακολουθεί το σύμβολο ίσον και εν συνεχεία τοποθετούνται αγκύλες, στις οποίες εμπεριέχεται η τιμή του λεξικού(όπως φαίνεται και στην σελίδα 81 του παραρτήματος). Για την δήλωση του περιεχομένου, δηλώνεται πρώτα το κλειδί μέσα σε διπλά εισαγωγικά. Ακολουθεί άνω κάτω τελεία και μετά το περιεχόμενο, το οποίο περικλείεται επίσης σε διπλά εισαγωγικά.

Το λεξικό που δημιουργήθηκε για τον συμβολομεταφραστή περιέχει τιμές από τον Πίνακα 4 και τον Πίνακα 5, δηλαδή περιέχει τις αντιστοιχίες του Assembly κώδικα και του Opcode, όπως και τα ονόματα των καταχωρητών σε αντιστοιχία με τις διευθύνσεις τους. Το λεξικό της παρούσης εργασίας παρατίθεται στο παρακάτω τμήμα κειμένου, όπου φαίνεται και η δομή που αναφέρθηκε παραπάνω.

```
mydict = {"movi": "000001", "mov": "000010", "mind": "000000", "ld": "000011",
"st": "000100", "add": "000101", "sub": "000110", "addc": "000111", "subc":
"001000", "addi": "001001", "subi": "001010", "and": "001011", "or": "001100",
"andi": "001101", "ori": "001110", "sl": "001111", "sr": "010000", "cmp":
"010001", "beq": "010010", "bgt": "010011", "jmp": "010100", "call": "010101",
"ret": "010110", "clr": "010111", "nop": "011000", "end": "111111", "W":
"00000","w": "00000", "status": "00001", "Status": "00001", "IC": "00010",
"PCL": "00011", "PCH": "00100", "Base": "00101", "reg1": "00110", "reg2":
"00111", "reg3": "01000", "reg4": "01001", "reg5": "01010", "reg6": "01011",
"reg7": "01100", "reg8": "01101", "reg9": "01110", "reg10": "01111", "reg11":
"10000", "reg12": "10001", "reg13": "10010", "reg14": "10011", "reg15": "10100",
"preg1": "10110", "preg2": "10111", "preg3": "11000", "preg4": "11001", "preg5":
"11010", "preg6": "11011", "preg7":"11100", "preg8": "11101", "preg9": "11110",
"preg10": "11111"}
```

Το λεξικό περιέχει στοιχεία από δύο πίνακες, έτσι χωρίζεται σε δύο μέρη όπως φαίνεται και χρωματικά. Το πρώτο μέρος περιέχει τα δεδομένα του Πίνακα 4 και ξεκινά από το πρώτο ζεύγος έως το προτελευταίο ζεύγος της έκτης γραμμής. Στο δεύτερο μέρος περιέχονται τα δεδομένα του Πίνακα 5, και αποτελείται από το τελευταίο ζεύγος της έκτης γραμμής έως το τελευταίο του λεξικού.

3.2.3 Βασικές μέθοδοι και συναρτήσεις

Για την παρούσα πτυχιακή εργασία, χρησιμοποιήθηκαν συγκεκριμένες μέθοδοι και συναρτήσεις, στις οποίες στηρίχτηκαν βασικά μέρη της λειτουργίας του κώδικα.

Η πιο βασική και χρήσιμη μέθοδος, η οποία χρησιμοποιήθηκε και πάνω σε αυτήν στηρίχτηκαν αρκετά μέρη της εργασίας, είναι η μέθοδος **«startswith»**. Η λειτουργία της είναι να ελέγχει αν κάποιο αλφαριθμητικό(string), ξεκινά με κάποια προκαθορισμένη τιμή που ορίζεται από τον προγραμματιστή. Αν ο έλεγχος είναι σωστός επιστρέφει σαν αποτέλεσμα 'True', εάν όχι επιστρέφει 'False'.

Με τη συγκεκριμένη μέθοδο δημιουργήθηκαν συναρτήσεις, οι οποίες στο πρόγραμμα του συμβολομεταφραστή ελέγχουν καταστάσεις δύο επιλογών. Μερικά παραδείγματα ακολουθούν παρακάτω.

Στην πρώτη περίπτωση, με την μέθοδο **«startswith»** βρίσκονται κάποιες από τις ειδικές κατηγορίες-περιπτώσεις των εντολών. Για παράδειγμα, αν μια εντολή ξεκινά με κάποια από τις εντολές Immediate, τότε επιστρέφει 'True'. Με ανάλογο τρόπο γίνεται ο έλεγχος της ύπαρξης του πεδίου shamt στις εντολές. Ελέγχεται, ακόμη, η ειδική κατηγορία του operand που μπορεί να είναι τύπου offset ή address. Μια ακόμα ειδική κατηγορία είναι αυτή που αποτελείται μόνο από μία εντολή assembly, δηλαδή το mnemonic. Τα συγκεκριμένα παραδείγματα αναφέρονται στο παράρτημα στις σελίδες 81 και 82.

Σε μια δεύτερη περίπτωση, γίνεται έλεγχος των διαφόρων αριθμητικών τύπων των ορισμάτων. Με τη χρήση της «startswith» πραγματοποιείται έλεγχος, με βάση τον Πίνακα 6, για το είδος του αριθμητικού ορίσματος. Εάν αρχίζει με «0x» είναι δεκαεξαδικός αριθμός, αν ξεκινά με «b» είναι δυαδικός και, τέλος αν αρχίζει από «d» είναι δεκαδικός αριθμός. Για παράδειγμα, όταν θέλουμε να φιλτράρουμε ένα όρισμα σε μια ειδική κατηγορία operand, το οποίο πρέπει να είναι σε δεκαδική μορφή, τότε ελέγχεται αν ξεκινά με το πρόθεμα «d». Το παράδειγμα που ακολουθεί είναι από το παράρτημα στην σελίδα 85.

```
elif(zz.startswith('b')):
```

Άλλη μια χρήσιμη μέθοδος είναι η **«replace»**, η οποία αντικαθιστά μια φράση που έχει οριστεί με μία άλλη, επίσης καθορισμένη. Η συγκεκριμένη μέθοδος χρησιμοποιήθηκε για να επιλυθεί ένα πρόβλημα του συμβολομεταφραστή που

προέκυπτε με τον χαρακτήρα «\n», ο οποίος ήταν ενσωματωμένος στο τελευταίο όρισμα στο τέλος κάθε γραμμής. Το σύμβολο αυτό, υποδηλώνει την αλλαγή γραμμής μέσα στο αρχείο κειμένου, κάτι το οποίο συμβαίνει καθώς γίνεται η χρήση του πλήκτρου «Enter». Αυτό είχε ως αποτέλεσμα κατά διαδικασία μετατροπής του κώδικα σε δυαδική μορφή να προκαλούνται σφάλματα. Για την επίλυση του συγκεκριμένου προβλήματος, έγινε η αντικατάσταση του χαρακτήρα αλλαγής γραμμής «\n», με κάποια κενά πριν από αυτόν. Ο λόγος της αντικατάστασης είναι πως το κενό ως διαχωριστικός χαρακτήρας δεν επηρεάζει πλέον τη λειτουργία του συμβολομεταφραστή. Έτσι, όταν διαβάζεται το αρχείο γραμμή προς γραμμή γίνεται η συγκεκριμένη αντικατάσταση. Στο παράδειγμα, στον κώδικα που ακολουθεί, η εντολή «replace» αντικαθιστά το πρώτο όρισμα, το οποίο βρίσκεται μέσα στην παρένθεση, δηλαδή αντικαθιστά το '\n' με το δεύτερο όρισμα που ακολουθεί μετά το κόμμα, το ' \n'.

```
x.write(line.replace('\n', '  \n'))
```

Η επόμενη μέθοδος είναι η **«split»**, η οποία διαχωρίζει μια συμβολοσειρά σε μια λίστα. Το διαχωριστικό σύμβολο μπορεί να οριστεί από το χρήστη. Το προεπιλεγμένο διαχωριστικό χαρακτήρων είναι το κενό διάστημα ανάμεσα σε συμβολοσειρές και αριθμούς. Η συγκεκριμένη μέθοδος χρησιμοποιείται από την βιβλιοθήκη «re». Για το λόγο αυτό, γράφεται πρώτα το όνομα της βιβλιοθήκης, ακολουθεί μια τελεία και μετά η μέθοδος. Στη δημιουργία του συμβολομεταφραστή χρησιμοποιήθηκε ως διαχωριστικό το κενό διάστημα αλλά και το κόμμα, όπου ξεχωρίζει το πρώτο με το δεύτερο operand.

```
y = re.split(" |,", line)
```

Μια ακόμη χρήσιμη μέθοδος είναι η **«zfill»**, η οποία προσθέτει μηδενικά ως χαρακτήρες στην αρχή της συμβολοσειράς, έως ότου φτάσει στο προκαθορισμένο μήκος που έχει οριστεί από τον προγραμματιστή. Εάν η τιμή της παραμέτρου ‘len’, δηλαδή το μήκος της συμβολοσειράς, είναι μικρότερο από το προκαθορισμένο, προσθέτει μηδενικά. Εάν το μήκος είναι μεγαλύτερο ή ίσο από αυτό που έχει οριστεί, τότε δεν προστίθεται κάτι. Η μέθοδος αυτή χρησιμοποιήθηκε για να καλυφθούν τα ελλείπει bit κατά την μετατροπή των δυαδικών ορισμάτων.

Τέλος, η μέθοδος **«append»** προσθέτει ένα στοιχείο στο τέλος της λίστας που έχει ορίσει ο προγραμματιστής. Για να μπορέσουν να συγκεντρωθούν οι εντολές στο τελικό δυαδικό αρχείο με τη σειρά, γίνεται η αποθήκευση τους σε λίστες. Η κάθε παράμετρος έχει τη δική της λίστα. Για παράδειγμα, μόλις γίνει η μετατροπή της πρώτης παραμέτρου της πρώτης εντολής, με τη χρήση της μεθόδου «append», τοποθετείται στην εκάστοτε λίστα. Εκεί προστίθενται με τη σειρά όλες οι πρώτες παράμετροι κάθε εντολής από του αρχείου κειμένου.

```
list1.append(y[0])
```

Οι κύριες συναρτήσεις που χρησιμοποιήθηκαν στην συγκεκριμένη πτυχιακή εργασία είναι δύο, η «zip» και η «len». Τα χαρακτηριστικά τους αναλύονται παρακάτω.

Η πρώτη σημαντική συνάρτηση είναι η «**zip**», η οποία όταν εφαρμόζεται σε δύο δεικτοδοτούμενα αντικείμενα(π.χ. λίστες) επιστρέφει ένα νέο δεικτοδοτούμενο αντικείμενο, κατάλληλα διαμορφωμένο. Το αντικείμενο αυτό αποτελεί μια διατρεχόμενη δομή(iterable) που αποτελείται από πλειάδες(tuples), όπου τα πρώτα στοιχεία σε κάθε λίστα συνδυάζονται μεταξύ τους. Στη συνέχεια, τα δεύτερα στοιχεία από την κάθε λίστα συνδυάζονται μεταξύ τους, κ.ο.κ. Η χρήση της συνάρτησης«zip» κατά τη δημιουργία του συμβολομεταφραστή κρίνεται πολύ βασική. Για παράδειγμα, για να γίνει έλεγχος εάν το πρώτο operand ανήκει σε κάποια ειδική κατηγορία, πρέπει να γνωρίζουμε εάν το mnemonic απαιτεί κάποια ειδική κατηγορία γι' αυτό. Έτσι, σε μια επαναληπτική δομή με τη χρήση της «zip» μπορούμε να διατρέχουμε δύο λίστες ταυτόχρονα: την πρώτη, αυτή του mnemonic,και τη δεύτερη, αυτή του πρώτου operand. Με την δυνατότητα αυτή υλοποιούνται όλοι οι έλεγχοι που απαιτούν οι ειδικές κατηγορίες των ορισμάτων.

```
for zz, mnemonic in zip(list2, list5):
```

Η δεύτερη συνάρτηση είναι η «**len**», η οποία επιστρέφει το μήκος των δεικτοδοτούμενων αντικειμένων. Στην περίπτωση που το αντικείμενο είναι μια συμβολοσειρά (string), η συνάρτηση «len» επιστρέφει τον αριθμό των χαρακτήρων που απαρτίζουν τη συμβολοσειρά. Καθώς όλο το πρόγραμμα του συμβολομεταφραστή βασίζεται στην επεξεργασία χαρακτήρων και συμβολοσειρών, το μήκος σε ειδικές περιπτώσεις ήταν μία παράμετρος προς έλεγχο, ώστε να διαπιστωθεί η ορθότητα του κώδικα που υποβάλει η χρήστης. Για παράδειγμα, εάν

μια εντολή έχει shamt,τότε το όρισμά του είναι δεκαδικός αριθμός και έχει όριο τιμών από το 0 έως το 7. Άρα, το shamt αποτελείται από δύο χαρακτήρες:

```
if (len(zz) <= 2):
```

ο πρώτος, είναι το πρόθεμα «d» και ο δεύτερος είναι ένας αριθμός. Το όρισμα πρέπει να είναι μικρότερο ή ίσο με δύο χαρακτήρες.

3.2.4 Επεξήγηση του κώδικα Assembler

Σε αυτό το υποκεφάλαιο γίνεται η επεξήγηση του κώδικα με τον οποίο υλοποιήθηκε ο συμβολομεταφραστής(assembler).

Αρχικά, ο κώδικας ξεκινά με μία συνάρτηση, η οποία περιέχει εντός της, όλες τις λειτουργίες του συμβολομεταφραστή. Ο λόγος που χρησιμοποιήθηκε η συγκεκριμένη προγραμματιστική τεχνική, είναι προκειμένου στη συνέχεια να κληθεί εύκολα από το γραφικό περιβάλλον που παρουσιάζεται παρακάτω.

Στη συνέχεια, εισάγονται οι απαραίτητες βιβλιοθήκες οι οποίες είναι δύο η «re» και η «sys» (για τη χρησιμότητα των οποίων βλ. σε παραπάνω υποκεφάλαιο 3.2.1). Έπειτα γίνεται η εισαγωγή ενός λεξικού, το οποίο περιέχει από τον Πίνακα 4 το register name σε αντιστοιχία με το address και από τον Πίνακα 5 τον κώδικα Assembly (mnemonic) σε αντιστοιχία με το opcode. Το λεξικό αυτό δημιουργήθηκε έτσι ώστε αν υπάρξει στο μέλλον κάποια αλλαγή, να γίνει διόρθωση άμεσα εντός του προγράμματος.

Ακολούθως γίνεται η δημιουργία και η δήλωση συναρτήσεων, οι οποίες πρόκειται να κληθούν στο κυρίως πρόγραμμα.

Η πρώτη συνάρτηση ελέγχει εάν κάποιος αριθμός είναι δυαδικός. Η συνάρτηση ονομάζεται **«isBinary»** και έχει ως όρισμα έναν αριθμό με τη μορφή αλφαριθμητικού. Η λειτουργία αυτή αποσκοπεί στον διεξοδικό έλεγχο του ορίσματος, ψηφίο προς ψηφίο, μέσω μίας δομής επανάληψης «for», η οποία ελέγχει εάν οι χαρακτήρες του αλφαριθμητικού αποτελούνται από 0 και 1. Εάν αποτελούνται, επιστρέφει ‘True’, σε διαφορετική περίπτωση επιστρέφει ‘False’. Ακολουθεί η συνάρτηση.

```
def isBinary(num: str):
    for i in num:
        if i not in ["0", "1"]:
            return False
    return True
```

Η δεύτερη συνάρτηση εξετάζει το ενδεχόμενο της ύπαρξης shamt σε μία εντολή. Οι εντολές που έχουν ως τέταρτο όρισμα shamt είναι δύο, η «sl» και η «sr». Με την συνθήκη ελέγχου «if» εξετάζεται το ενδεχόμενο της έναρξης του mnemonic με κάποια από τις δύο εντολές. Στην περίπτωση που το mnemonic ξεκινά όντως με κάποια από τις δύο εντολές, επιστρέφει 'True', σε διαφορετική περίπτωση εάν το mnemonic ξεκινά με εντολές Immediate ή τύπου Branch επιστρέφει 'False' και στη θέση του shamt ο χώρος παραμένει κενός έτσι ώστε να χρησιμοποιηθεί από το literal των εντολών αυτών, που απαιτεί 8 bit.

Οι επόμενες συναρτήσεις ελέγχουν εάν μία εντολή ανήκει σε κάποια από τις ειδικές κατηγορίες βασιζόμενες στο mnemonic της. Ειδικές θεωρούνται οι κατηγορίες των εντολών Immediate, εντολών που το πρώτο operand έχει ως όρισμα offset ή address. Ακόμη μια ειδική κατηγορία είναι η τελευταία συνάρτηση, η οποία ελέγχει εάν η εντολή αποτελείται μόνο από mnemonic. Όλες οι παραπάνω συναρτήσεις, είναι δομημένες με τον ίδιο τρόπο. Εξετάζουν το ενδεχόμενο εάν η εντολή ξεκινά με κάποιο από τα ανάλογα mnemonic σε κάθε ειδική κατηγορία. Τέλος, αν ισχύει ή όχι η συνθήκη επιστρέφουν 'True' ή 'False'. Ακολουθεί η τελευταία συνάρτηση, η οποία ελέγχει την ύπαρξη εντολής που αποτελείται μόνον από mnemonic. Με ανάλογο τρόπο γράφονται και οι υπόλοιπες συναρτήσεις.

```
def onlymnemonic(onlymnem):
    pref = ["ret", "nop", "end"]
    if onlymnem.startswith(tuple(pref)):
        return True
    return False
```

Στην συνέχεια το πρόγραμμα δέχεται ως είσοδο ένα αρχείο κειμένου με όνομα «temp» και ανοίγοντας το, με την ιδιότητα της ανάγνωσης, περιέχει τις εντολές σε γλώσσα assembly που έχει γράψει ο χρήστης.

Η επόμενη ενέργεια είναι η αντιγραφή του αρχείου «temp.txt» σε ένα νέο αρχείο, με όνομα «out». Ο λόγος της δημιουργίας του νέου αρχείου είναι η αντικατάσταση του χαρακτήρα ‘ \n’ με την μέθοδο «replace». Στην αντικατάσταση προστίθενται κάποια κενά πριν από αυτόν, με σκοπό ο επόμενος χαρακτήρας, μετά τον τελευταίο κάθε γραμμής, να είναι το κενό. Ο χαρακτήρας αλλαγής γραμμής, δηλαδή το ‘\n’ δημιουργούσε πρόβλημα κατά την μετατροπή του κώδικα σε δυαδική μορφή.

Ακολουθούν οι δηλώσεις με κενές λίστες, στις οποίες καταχωρούνται σημαντικά στοιχεία. Για την δήλωση μίας κενής λίστας, δίνεται το όνομα της, ακολουθεί το σύμβολο ίσον κι έπειτα δύο αγκύλες χωρίς να περιέχουν κάτι και χωρίς κενό ανάμεσά τους. Οι συγκεκριμένες λίστες έχουν σκοπό την αποθήκευση δεδομένων από το αρχείο εισόδου και δεδομένα για το αρχείο εξόδου σε δυαδική μορφή.

Η λειτουργία από τις πρώτες τέσσερις λίστες, αφορά την αποθήκευση στοιχείων από το αρχείο εισόδου «temp»: το mnemonic τοποθετείται στη λίστα list1, το πρώτο operand στη λίστα list2, το δεύτερο operand στη λίστα list3 και τέλος το shamt τοποθετείται στη λίστα list3i.

Στις επόμενες λίστες καταχωρούνται τα δυαδικά αποτελέσματα μετά από την μετατροπή του κώδικα assembly. Στη list5 αποθηκεύεται το mnemonic ενώ στη list4 το opcode. Το function bit αποθηκεύεται στη λίστα fbit. Το αποτέλεσμα του πρώτου operand σε δυαδική μορφή αποθηκεύεται στη λίστα listop1 και τα κλειδιά από το πρώτο operand στη λίστα listoop1. Αντίστοιχα, το δεύτερο operand σε δυαδική μορφή αποθηκεύεται στη listop2 και το κλειδί από το δεύτερο operand στη listoop2. Τέλος, στη λίστα listop3 αποθηκεύεται το shamt μετά την μετατροπή του σε δυαδική μορφή. Στον παρακάτω Πίνακα 7 βρίσκονται όλες οι λίστες σε αντιστοιχία με τα δεδομένα που αποθηκεύονται σε αυτές(όπως φαίνεται και στην σελίδα 83 του παραρτήματος).

Πίνακας 7 - Αντιστοιχία Λίστας με Δεδομένα Αποθήκευσης

Κενές λίστες	Δεδομένα Αποθήκευσης
list1	Mnemonic
list2	Πρώτο operand
list3	Δεύτερο operand
list3i	shamt
list4	B* Opcode

list5	Mnemonic
fbit	B* Function bit
listop1	B* Πρώτο operand
listoop1	B* Κλειδί από το Πρώτο operand
listop2	B* Δεύτερο operand
listoop2	B* Κλειδί από το Δεύτερο operand
listop3	B* shamt

Ο συμβολισμός B* είναι για τα δεδομένα μετά την μετατροπή σε δυαδική μορφή

Κατά τη διαδικασία της συγγραφής των εντολών assembly, ίσως προκύψουν κάποια συντακτικά λάθη στον κώδικα. Για τον λόγο αυτό, είναι αναγκαία η δημιουργία ενός νέου αρχείου, στο οποίο θα αποθηκεύονται όλα τα λάθη που βρέθηκαν. Το αρχείο κειμένου ονομάζεται «Error_output». Η δημιουργία του πραγματοποιείται στον ίδιο φάκελο με το αρχείο της Python και του αρχείου κειμένου εισόδου, μόλις τρέξει το πρόγραμμα του συμβολομεταφραστή.

3.2.5 Κυρίως πρόγραμμα και μετατροπή δεδομένων

Εφόσον έχουν γίνει οι απαραίτητες δηλώσεις και αρχικοποιήσεις, συνεχίζουμε στο κυρίως πρόγραμμα, όπου γίνεται η μετατροπή του κώδικα. Αρχικά, ανοίγοντας το αρχείο «out» με την εντολή open, διαβάζονται οι γραμμές του αρχείου μία προς μία. Έπειτα, με τη μέθοδο split διαχωρίζονται οι εντολές ανά κατηγορία και αποθηκεύονται σε τέσσερις λίστες. Τα στοιχεία που χρησιμοποιεί η split ως διαχωριστικό των εντολών, είναι το κενό ανάμεσα σε δύο λέξεις και το σύμβολο κόμμα. Το κόμμα υπάρχει αμέσως μετά το πρώτο operand και ανάμεσα στο δεύτερο operand με το shamt, στην περίπτωση που η εντολή ανήκει σε ειδική κατηγορία και έχει shamt. Οι λειτουργίες αυτές περιγράφονται στις παρακάτω γραμμές κώδικα.

```
x = open('out.txt')
    for line in x:
        y = re.split(" |,", line)
        list1.append(y[0])
        list2.append(y[1])
        list3.append(y[3])
        list3i.append(y[5])
```

Εύρεση Mnemonic

Η πρώτη μετατροπή που πραγματοποιείται είναι αυτή του mnemonic. Στην συγκεκριμένη περίπτωση υπάρχει μια αντιστοιχία του mnemonic με το opcode και όχι μετατροπή. Η αντιστοιχία βρίσκεται εφόσον υπάρχει προκαθορισμένο σύνολο ζευγαριών στο λεξικό, από τον Πίνακα 4. Έτσι ελέγχεται αρχικά εάν το mnemonic που δόθηκε ανήκει στο λεξικό. Στην περίπτωση που ανήκει, αποθηκεύεται το κλειδί-mnemonic στη list4 και η τιμή-opcode στη list5. Υπάρχουν δύο περιπτώσεις σφαλμάτων που μπορεί να συμβούν σε αυτή την αντιστοιχία: πρώτον, το mnemonic να είναι κενό, που σημαίνει πως ο κέρσορας βρίσκεται σε μία νέα γραμμή, η οποία δεν περιέχει εντολές και δεύτερον, η περίπτωση η εντολή να μην ανήκει στο λεξικό. Για κάθε σφάλμα υπάρχει σχετικό μήνυμα, με περιεχόμενο τον αριθμό γραμμής που συνέβη το σφάλμα, το οποίο αποθηκεύεται στο αρχείο κειμένου Error_output. Ο κώδικας για την μετατροπή αυτή βρίσκεται στη σελίδα 84 του παραρτήματος.

Εύρεση πρώτου operand

Αμέσως μετά γίνεται η μετατροπή του πρώτου operand σε δυαδική μορφή. Για να υλοποιηθεί αυτό είναι αναγκαίο να προσπελάζονται δύο λίστες ταυτόχρονα, διότι οι επιτρεπτοί τύποι ορισμάτων του operand εξαρτώνται από την προηγούμενη εντολή, το mnemonic. Η παράλληλη προσπέλαση σε δύο λίστες, γίνεται με την χρήση της μεθόδου «zip». Η πρώτη λίστα είναι η list2, που περιέχει το πρώτο operand και η list5 που περιέχει το κλειδί-mnemonic. Στην περίπτωση του πρώτου operand υπάρχουν πέντε υποπεριπτώσεις προς έλεγχο στις οποίες μπορεί να ανήκει.

Στην πρώτη περίπτωση εξετάζεται το ενδεχόμενο η εντολή να μην περιέχει κάποιο operand, αλλά να αποτελείται μόνο από mnemonic. Η υλοποίηση του συγκεκριμένου ελέγχου γίνεται μέσω της συνάρτησης «onlymnemonic», που ελέγχει εάν το mnemonic ανήκει στις τρείς εντολές «call», «ret» και «end». Ένας επόμενος έλεγχος είναι το μήκος της εντολής να είναι μικρότερο ή ίσο με ένα χαρακτήρα. Τότε στη θέση της λίστας «listop1» για το πρώτο operand προσθέτονται πέντε μηδενικά «00000», που δηλώνουν την μη ύπαρξη του, στην συγκεκριμένη εντολή. Εάν το μήκος του operand είναι μεγαλύτερο από ένα, σε εντολή που αποτελείται μόνο από mnemonic, τότε στο αρχείο αποθήκευσης σφαλμάτων εμφανίζεται σχετικό μήνυμα και στη θέση της λίστας «listop1» εμφανίζεται ο χαρακτήρας «*» που δηλώνει το σφάλμα.

Στην δεύτερη περίπτωση εξετάζεται το ενδεχόμενο το operand να έχει τη μορφή address. Οι τιμές που μπορεί να πάρει στην κατηγορία αυτή, εξαρτώνται από το mnemonic της εντολής. Ο έλεγχος πραγματοποιείται με τη συνάρτηση «address», η οποία εξετάζει εάν το mnemonic της εντολής είναι η εντολή «jmp» ή η εντολή «call». Οι τιμές σε αυτή την περίπτωση είναι δυαδικές και ξεκινούν με το πρόθεμα «b». Με σκοπό την εξακρίβωση πως ο αριθμός αυτός είναι όντως δυαδικός, είναι αναγκαίο να μείνει ο αριθμός χωρίς το πρόθεμα του. Έτσι γίνεται η αποθήκευση του σε μια νέα μεταβλητή, και ο αριθμός διαβάζεται μετά τον πρώτο χαρακτήρα κι έπειτα. Εφόσον έχει αποθηκευτεί ο αριθμός στη νέα μεταβλητή, με τη συνάρτηση «isBinary» εξακριβώνεται εάν το address αποτελείται μόνο από 0 και 1. Όταν η συνάρτηση επιστρέφει 'True' πραγματοποιείται ακόμη ένας έλεγχος για το μήκος του address, με τη συνάρτηση len, η οποία επιστρέφει το μήκος εάν είναι ίσο με δέκα ψηφία. Εάν είναι όντως δέκα, τότε συμπληρώνονται στην αρχή με τη μέθοδο «zfill», τρία μηδενικά που χρειάζονται για να φτάσει το address να αποτελείται από τα απαραίτητα ψηφία. Τα μηδενικά στην αρχή του αριθμού, δεν επηρεάζουν τον αριθμό. Μόλις ολοκληρωθεί αυτή η διαδικασία αποθηκεύεται το αποτέλεσμα στη «listop1». Υπάρχουν όμως και κάποιες περιπτώσεις σφαλμάτων, όπως το μεγαλύτερο μέγεθος του address, ο λάθος αριθμητικός τύπος του address και η περίπτωση που το operand δεν ξεκινάει με το σωστό πρόθεμα «b». Σε όλες τις παραπάνω περιπτώσεις εμφανίζεται στην «listop1» το σύμβολο που δηλώνει σφάλμα «*».

Στην τρίτη περίπτωση εξετάζεται η ειδική κατηγορία του πρώτου operand να είναι της μορφής offset. Οι επιτρεπόμενες τιμές για τη συγκεκριμένη κατηγορία είναι οι δεκαδικοί αριθμοί, με την ιδιαιτερότητα πως είναι προσημασμένοι. Η ειδοποιός διαφορά των προσημασμένων αριθμών σε σχέση με τους μη προσημασμένους είναι στο πρόθεμα τους. Οι θετικοί αριθμοί ξεκινούν με «d+» ενώ οι αρνητικοί με «d-».

Η πρώτη κατηγορία ξεκινά με τον έλεγχο της συνάρτησης «offset», η οποία εξετάζει εάν το mnemonic είναι κάποιο από τις εντολές τύπου Branch όπως η «beq» ή η «bgt». Στην περίπτωση που η συνάρτηση επιστρέψει 'True' συνεχίζεται ο έλεγχος με τη μέθοδο «startswith», η οποία εξετάζει, αρχικά το πρόθεμα εάν είναι θετικό, δηλαδή το «d+». Στην συνέχεια, με τη συνάρτηση len ελέγχεται το μήκος της εντολής εάν είναι μικρότερο ή ίσο από πέντε χαρακτήρες. Το ανώτατο όριο είναι πέντε χαρακτήρες, διότι οι πρώτοι δύο αποτελούν το πρόθεμα και οι άλλοι τρείς χαρακτήρες τον δεκαδικό αριθμό. Στην περίπτωση που το offset είναι εντός του

ορίου, διαβάζεται η εντολή από το τρίτο χαρακτήρα και μετά, δηλαδή μόνο το μέρος του αριθμού και αποθηκεύεται σε μια μεταβλητή. Ακολουθεί άλλος ένας έλεγχος για τον εύρος τιμών του αριθμού που είναι από το 0 έως το +127, θεωρώντας πως το 0 ανήκει στους θετικούς αριθμούς. Τέλος, εάν το πλήθος των ψηφίων και το εύρος τιμών είναι εντός του επιτρεπόμενου, μετατρέπεται ο δεκαδικός αριθμός σε δυαδικό με μία συνάρτηση και το αποτέλεσμα αποθηκεύεται στη λίστα «listop1». Η μετατροπή και αποθήκευση του offset παραθέτονται μαζί την επεξήγηση των εντολών.

```
result = "{0:08b}".format(res2, 10)
 listop1.append(result)
```

Η μετατροπή αυτή πραγματοποιείται με τη μέθοδο «**format**», η οποία μορφοποιεί τις καθορισμένες τιμές και τις εισάγει μέσα στο σύμβολο κράτησης θέσης της συμβολοσειράς. Το σύμβολο κράτησης θέσης ορίζεται χρησιμοποιώντας τις αγκύλες. Η συγκεκριμένη μέθοδος επιστρέφει τη μορφοποιημένη συμβολοσειρά. Η πρώτη εντολή, στο δεξί μέρος της ισότητας περιέχει μέσα στις αγκύλες το σύμβολο κράτησης θέσης(placeholder) και έχει τη μορφή {0:08b}, που δηλώνει τη μετατροπή στο δυαδικό σύστημα ενός αριθμού των 8 bit. Ακολουθεί η συνάρτηση format με τις καθορισμένες τιμές μέσα στην παρένθεση. Η πρώτη τιμή-μεταβλητή «res2» περιέχει τον αριθμό του offset. Ο αριθμός 10 που ακολουθεί, είναι το αριθμητικό σύστημα που βρίσκεται η τιμή, στην προηγούμενη μεταβλητή «res2». Το αποτέλεσμα της μετατροπής τοποθετείται σε μια μεταβλητή με το όνομα result. Στην δεύτερη εντολή με τη μέθοδο append αποθηκεύεται το αποτέλεσμα της μετατροπής στην λίστα «listop1».

Οι περιπτώσεις σφαλμάτων που μπορεί να προκύψουν κατά τη συγγραφή του κώδικα για το offset είναι δύο. Στην πρώτη περίπτωση αν ο δεκαδικός αριθμός είναι μεγαλύτερος του 127 και στη δεύτερη αν περιέχει περισσότερα από τρία ψηφία. Σε κάθε περίπτωση εμφανίζεται το σχετικό μήνυμα, καθώς και η γραμμή στην οποία συνέβη το σφάλμα. Τέλος, στη λίστα «listop1» αποθηκεύεται το «*» σαν ένδειξη λάθους.

Παρόμοια διαδικασία ελέγχου πραγματοποιείται και για τους αρνητικούς δεκαδικούς αριθμούς offset. Αρχικά, το offset ελέγχεται με τη μέθοδο «startswith», εάν ξεκινά με

το πρόθεμα «d-». Με την συνάρτηση len εξετάζεται το μήκος του offset, το οποίο πρέπει να είναι μικρότερο ή ίσο με πέντε χαρακτήρες. Όπως στο θετικό, έτσι και στο αρνητικό offset, οι πρώτοι δύο χαρακτήρες είναι το πρόθεμα «d-» και οι υπόλοιποι μπορεί να αποτελούν τον αριθμό. Το όριο των αριθμών μπορεί να είναι από το -1 έως το -128. Από το σημείο αυτό κι έπειτα, υπάρχει μία διαφοροποίηση με το θετικό offset. Πριν γίνει η μετατροπή του σε δυαδική μορφή, για να βρεθεί ο αριθμός εφαρμόζεται η μέθοδος συμπληρώματος ως προς 2. Στη μέθοδο αυτή αφαιρείται από τον αριθμό 256, ο αρνητικός προσημασμένος αριθμός και το υπόλοιπο που προκύπτει αποθηκεύεται σε μία μεταβλητή με το όνομα «m», όπως υπάρχει στο παράρτημα στις σελίδες 86 και 87. Στη συνέχεια πραγματοποιείται η διαδικασία μετατροπής του σε δυαδικό αριθμό. Το αποτέλεσμα αποθηκεύεται στη λίστα «listop1». Όπως και στο θετικό offset, έτσι και στο αρνητικό, υπάρχουν οι ίδιοι δύο τύποι σφαλμάτων. Οι περιπτώσεις είναι δύο ο αριθμός να είναι μεγαλύτερος του +127 ή του -128, ή να ξεπερνά το όριο των τριών ψηφίων. Στο αρχείο «Error_output» εμφανίζεται στην κάθε περίπτωση το ανάλογο μήνυμα όπως και στη λίστα «listop1» το «*».

Στην τέταρτη περίπτωση εξετάζεται το ενδεχόμενο πριν το πρώτο operand να προηγείται εντολή mnemonic, η οποία ανήκει στην ειδική κατηγορία Immediate. Ο έλεγχος αυτός γίνεται μέσω της συνάρτησης «mnemim» που συγκρίνει εάν η εντολή mnemonic ξεκινά με κάποια από τις εντολές «movi», «addi», «subi», «andi» και «ori». Οι τιμές που μπορεί να πάρει το operand σε αυτή την κατηγορία, είναι τιμές από το λεξικό που προήλθαν από τον Πίνακα 5, δυαδικές και δεκαεξαδικές τιμές.

Πραγματοποιείται έλεγχος εάν το operand υπάρχει και ταυτίζεται με κάποιο κλειδί από το λεξικό. Εάν βρεθεί το operand στο λεξικό, γίνεται η αντιστοιχία ανάμεσα στο κλειδί και την τιμή της διεύθυνσης, η οποία αποθηκεύεται στη λίστα «listop1».

Στη δεύτερη υποπερίπτωση το πρώτο operand μπορεί να είναι δυαδικός αριθμός. Ξεκινώντας ελέγχεται το πρόθεμα εάν είναι «b». Εν συνεχεία διαβάζεται ο αριθμός μετά τον πρώτο χαρακτήρα, που αποτελεί το πρόθεμα. Με τη συνάρτηση «isBinary» ελέγχονται τα ψηφία του αριθμού εάν είναι 0 ή 1 και αν επιστρέψει ‘True’ με τη συνάρτηση len συμπληρώνονται τα ελλείπει ψηφία, ώστε να φτάσουν τα πέντε στο σύνολο. Τα ψηφία που λείπουν καλύπτονται από τα αριστερά του αριθμού με μηδενικά, τα οποία δεν αλλάζουν τον αριθμό.

Στην τελευταία υποπερίπτωση το πρώτο operand μπορεί να έχει ως αριθμητικό τύπο δεκαεξαδικούς αριθμούς. Με την μέθοδο «startswith» πραγματοποιείται έλεγχος για το πρόθεμα των δεκαεξαδικών που είναι το «0x». Τοποθετώντας το μήκος του operand σε μια μεταβλητή, ελέγχεται εάν το μήκος των χαρακτήρων είναι μικρότερο ή ίσο των τεσσάρων. Αυτή η ενέργεια συμβαίνει, διότι το όριο των δεκαεξαδικών αριθμών είναι τα δύο ψηφία, καθώς τα πρώτα δύο τα καταλαμβάνει το πρόθεμα. Το εύρος των τιμών ξεκινά από το 00 και καταλήγει στο 1F. Έτσι ο δεκαεξαδικός μετατρέπεται σε δεκαδικό για να ελεγχθεί το εύρος εάν είναι εντός ορίων και αν είναι έπειτα, γίνεται η μετατροπή του σε δυαδική μορφή με τη συνάρτηση «format». όπως παρουσιάζεται στο παράρτημα στη σελίδα 87.

```
res = "{0:05b}".format(int(zz, 16))
```

Στην καθορισμένη τιμή μέσα σε αγκύλες υπάρχει το πλήθος των δυαδικών ψηφίων που είναι το ‘08b’, ενώ τιμή της κράτησης θέσης τοποθετείται η μεταβλητή με τον δεκαεξαδικό αριθμό και ακολουθεί μετά το κόμμα ο παρόν αριθμητικός τύπος, δηλαδή το 16. Μετατρέποντας το αποτέλεσμα της προηγούμενης μετατροπής σε μορφή «string», τοποθετείται σε μια νέα μεταβλητή η δυαδική μορφή από τον τέταρτο χαρακτήρα έως τον όγδοο, έτσι ώστε να γίνει 5 bit . Το αποτέλεσμα αυτό αποθηκεύεται στη λίστα «listop1».

Εφόσον υπάρχουν περισσότερες περιπτώσεις σε αυτή την κατηγορία υπάρχουν και περισσότερες πιθανότητες σφαλμάτων. Για παράδειγμα, το λάθος μέγεθος του δυαδικού αριθμού, ή ο αριθμητικός τύπος να μην είναι δυαδικός, ή ο δεκαεξαδικός αριθμός να υπερβαίνει τα δύο ψηφία και τέλος να είναι λάθος ο τύπος του αριθμού.

Στην πέμπτη περίπτωση το mnemonic δεν ανήκει σε καμία από τις παραπάνω ειδικές κατηγορίες. Με τον άρση του περιορισμού των ειδικών περιπτώσεων, το πρώτο operand μπορεί να πάρει τιμές από το λεξικό, ή οι διευθύνσεις να παριστάνονται με δεκαεξαδικές, δυαδικές ή δεκαδικές τιμές. Παραθέτονται τα τμήματα κώδικα για τον κάθε έλεγχο ξεχωριστά.

Η πρώτη αναζήτηση αποσκοπεί στην εύρεση του πρώτου operand μέσω της αναζήτησης του στο λεξικό. Εάν βρεθεί, με τη χρήση του βρόγχου ‘for’ γίνεται η αντιστοιχία ανάμεσα στο κλειδί και την τιμή. Έπειτα αποθηκεύεται το κλειδί στη λίστα «listoop1» και η τιμή της διεύθυνσης στη λίστα «listop1».

```
if zz in mydict:
        for kk, vv in mydict.items():
         if kk == zz:
           listoop1.append(kk)
           listop1.append(vv)
```

Η δεύτερη αναζήτηση εστιάζει στην εύρεση του πρώτου operand, εξετάζοντας το ενδεχόμενο να είναι σε δεκαεξαδική μορφή. Αρχικά, ελέγχεται το πρόθεμα του operand με στην μέθοδο «startswith», εάν ξεκινά με «0x». Έπειτα, υπάρχει περιορισμός για το μήκους, που επιτρέπονται έως δύο δεκαεξαδικά ψηφία, εφόσον τα δύο αποτελούν το πρόθεμα. Ακολουθεί η μετατροπή σε δυαδική μορφή, που το αποτέλεσμά της αποθηκεύεται στη λίστα «listop1» η μεταβλητή με το όνομα «aa» που περιέχει το αποτέλεσμα της μετατροπής από το τέταρτο έως τον όγδοο χαρακτήρα. Τέλος, εάν υπάρχει κάποιο λάθος ακολουθεί το μήνυμα σφάλματος, αποθηκεύοντας στη λίστα «listop1» το σύμβολο «*».

```
    elif (zz.startswith('0x')):
      if  (len(zz) <= 4):
        res = (str(zz))
        res1 = (res[2:])
        res2 = (str(res1))
        res3 = int(res2,16)
        if res3 <= 31:
          res4 = "{0:05b}".format(int(res2, 16))
          listop1.append(res4)
        else:
          print("Error ! The code has more than 1f as a hexadecimal number as
              first operand in line : ", counter, file=Error)
          listop1.append("*")
      else:
        print("Error ! The code has more than 2 digits as a hexadecimal
            number as first operand in line : ", counter, file=Error)
        listop1.append("*")
```

Η τρίτη αναζήτηση αφορά την περίπτωση που το operand ανήκει στους δεκαδικούς αριθμούς. Ο πρώτος έλεγχος εξετάζει το πρόθεμα, το οποίο σε αυτή την περίπτωση είναι το «d». Στη συνέχεια, εξετάζεται το μήκος το οποίο ενδείκνυται να είναι μικρότερο από τρείς χαρακτήρες με εύρος τιμών από το 0 έως τον αριθμό 31. Ακολουθεί η μετατροπή σε δυαδική μορφή αποθηκεύοντας το αποτέλεσμα στη λίστα «listop1». Τέλος, ακολουθούν τα μηνύματα σφαλμάτων, όπου το πρώτο αφορά το ανώτατο όριο αριθμού και το δεύτερο το λάθος μέγεθος του.

```
elif(zz.startswith('d')):
        if (len(zz)<=3):
          res = (str(zz))
          res1 = (res[1:])
          res2 = (int(res1))
          if res2 <= 31:
             result = "{0:0b}".format(int(res1, 10))
             listop1.append(result)
           else:
             print("Error!The decimal number is out of boundaries in line: "
                   ,counter, file=Error)
             listop1.append("*")
           else:
             print("Error ! Wrong decimal size of the first operand in line : ",
               counter, file=Error)
             listop1.append("*")
```

Στην προτελευταία εκδοχή η κατηγορία του πρώτου operand είναι σε δυαδική μορφή. Η διαδικασία για την αποθήκευση του τελικού αποτελέσματος είναι ίδια με όσες είχαν τον ίδιο τύπο αριθμού. Ξεκινώντας με τον έλεγχο για το πρόθεμα που είναι το «b», συνεχίζοντας με τον έλεγχο κάθε χαρακτήρα για να εξακριβωθεί εάν αποτελείται από 0 και 1. Τέλος, αποθηκεύεται το αποτέλεσμα στη λίστα «listop1» και ακολουθούν τα μηνύματα σφαλμάτων.

```
elif(zz.startswith('b')):
        res = (str(zz))
        res1 = (zz[1:])
        if isBinary(res1) == True:
          a = len(res1)
          if (a <= 5):
            f = (5-a)
            ff = (res1.zfill(f+a))
            listop1.append(ff)
          else:
              print("Error ! Wrong size of the first operand in line : ", counter,
                    file=Error)
              listop1.append("*")
        else:
           print("Error ! Wrong type of number!!! Number of the first operand isn't
                 binary in line : ", counter, file=Error)
           listop1.append("*")
```

Η τελευταία κατηγορία που είναι απαραίτητο να εξεταστεί είναι αυτή, που το operand που δεν υπάρχει. Εάν είναι κενό το μέρος τοποθέτησης του στοιχείου operand, τότε στη λίστα «listop1» τοποθετούνται πέντε μηδενικά «00000». Αυτό συμβαίνει έτσι ώστε να συμπληρωθούν τα πέντε bit του πρώτου operand.

```
elif (zz.startswith("")):
        listop1.append("00000")
```

Εύρεση δεύτερου operand

Η διαδικασία για την εύρεση του δεύτερου operand είναι ανάλογη του πρώτου, με κάποιες μικρές διαφοροποιήσεις. Αρχικά, γίνεται προσπέλαση σε δύο λίστες ταυτόχρονα, διότι οι επιτρεπτοί τύποι ορισμάτων του operand εξαρτώνται από την εντολή mnemonic. Η προσπέλαση στις δύο λίστες γίνεται με την χρήση της μεθόδου «zip». Η πρώτη λίστα είναι η list3, η οποία περιέχει το δεύτερο operand που αποθηκεύτηκε από το αρχείο εισόδου και η list5 που περιέχει το κλειδί-mnemonic του αντίστοιχου operand. Στην περίπτωση του δεύτερου operand υπάρχουν δύο κατηγορίες οι οποίες θα εξεταστούν.

Η πρώτη περίπτωση, στην οποία μπορεί να ανήκει το δεύτερο operand με βάση το mnemonic, είναι η κατηγορία των εντολών Immediate. Ο έλεγχος πραγματοποιείται μέσω της συνάρτησης «mnemim», η οποία συγκρίνει εάν η εντολή mnemonic ξεκινά με κάποια από τις εντολές που έχουν ως τελευταίο χαρακτήρα το ‘i’. Το σύνολο των εντολών είναι οι εξής: «movi», «addi», «subi», «andi» και «ori». Εάν όντως το mnemonic ανήκει σε κάποια από τις εντολές αυτές, το operand περιορίζεται στις δυαδικές, δεκαεξαδικές και δεκαεξαδικές τιμές των 8 bit(3 bit του shamt και 5 του operand). Ο έλεγχος που ακολουθεί είναι πανομοιότυπος με αυτόν του πρώτου operand, με μόνη διαφορά τη λίστα αποθήκευσης, η οποία είναι η «listop2».

Η δεύτερη περίπτωση είναι αυτή, η οποία έχει ως προηγούμενη εντολή το address, έτσι ο χώρος του δευτέρου operand παραμένει κενός και δεν γεμίζει με μηδικά, έτσι ώστε να φιλοξενήσει τα 5 από τα 10 bit του address.

Στην περίπτωση του δεύτερου operand μπορεί να πάρει ως όρισμα όλες τις τιμές ανεξάρτητος του mnemonic που προηγείται. Οι τιμές μπορεί να είναι από το λεξικό ή δεκαεξαδικές, δυαδικές και δεκαδικές τιμές. Αυτή η περίπτωση διαφοροποιείται σε ένα σημείο σε σχέση με το πρώτο operand, αυτό είναι η λίστα αποθήκευσης των αποτελεσμάτων σε δυαδική μορφή που είναι η «listop2». Ο έλεγχος για την εύρεση του δεύτερου operand βρίσκεται στις σελίδες 90-93 του παραρτήματος.

Εύρεση shamt

Η εντολή shamt είναι ένας δεκαδικός αριθμός, ο οποίος ακολουθεί μετά το δεύτερο operand σε ειδικές περιπτώσεις. Η ύπαρξη shamt προϋποθέτει να προηγείται συγκεκριμένο mnemonic, πιο συγκεκριμένα μία από τις δύο εντολές την «sl» ή την

«sr». Με τη χρήση ενός βρόχου for και της μεθόδου «zip» διατρέχονται ταυτόχρονα οι δύο ενδιαφερόμενες λίστες: η «list3i», η οποία περιέχει το shamt και η «list5», που περιέχει το mnemonic. Στη συνέχεια, δημιουργείται μια νέα λίστα με το όνομα «prefixes», που περιέχει όλες τις πιθανές περιπτώσεις του shamt. Οι περιπτώσεις αυτές είναι δεκαδικοί αριθμοί από το 0 έως το 7 με πρόθεμα το «d». Ακολούθως, με τη μέθοδο «startswith» ελέγχεται εάν το mnemonic ξεκινά με κάποια από τις δύο εντολές. Εάν η συνάρτηση επιστρέψει 'True', ελέγχεται το περιεχόμενο του shamt εάν ανήκει στη λίστα «prefixes». Ακολουθεί έλεγχος με τη συνάρτηση «len», που ελέγχει το μήκος της εντολής, το οποίο πρέπει να είναι μικρότερο ή ίσο με δύο χαρακτήρες. Σε μια μεταβλητή με όνομα «res1» αποθηκεύεται ο δεύτερος χαρακτήρας, που είναι μόνο ο αριθμός, χωρίς το πρόθεμα. Έτσι με τη συνάρτηση «format» γίνεται η μετατροπή του δεκαδικού σε δυαδικό αριθμό, καθώς το αποτέλεσμα αποθηκεύεται στη λίστα «listop3». Ακολουθούν τα μηνύματα των σφαλμάτων. Οι περιπτώσεις σφαλμάτων είναι δύο: το λάθος μέγεθος του shamt και το mnemonic που ελέγχεται να μην ανήκει στην κατηγορία που περιέχει shamt. Σε αυτές τις δύο περιπτώσεις στη λίστα «listop3» εμφανίζεται το «*».Εάν το mnemonic ανήκει στις εντολές Immediate ή είναι beq ή bgt, στη λίστα «listop3» δεν αποθηκεύεται τίποτα. Στον τελευταίο έλεγχο που ανήκουν οι περισσότερες εντολές, οι οποίες δεν περιέχουν shamt. Σε αυτή την περίπτωση στη λίστα «listop3», αποθηκεύονται τρία μηδενικά «000». Ο έλεγχος για την εύρεση του shamt βρίσκεται στις σελίδες 93-94 του παραρτήματος.

Εύρεση Function bit

Τελευταία προσθήκη πριν την ολοκλήρωση του συμβολομεταφραστή είναι η προσθήκη του function bit. Σε αυτό το τμήμα της εντολής δεν υπάρχει κάποια μετατροπή, καθώς ο αριθμός αποτελείται από 0 ή 1. Μια ακόμη διαφορά με τα υπόλοιπα τμήματα της κάθε εντολής, είναι πως το function bit δεν το γράφει ο χρήστης, αλλά καθορίζεται από στο mnemonic και προσθέτεται στο τελικό αρχείο αυτόματα, μέσω του κώδικα.

Η κατανομή πραγματοποιείται με μια δομή επανάληψης for, με την οποία γίνεται η προσπέλαση της λίστας «list5», η οποία περιέχει το mnemonic. Για το σκοπό αυτό έχουν δημιουργηθεί τρείς διαφορετικές λίστες. Η πρώτη ονομάζεται «pref1» και περιέχει τις εντολές cmp, beq και bgt και η δεύτερη ονομάζεται «pref0», η οποία

περιέχει τις εντολές movi, mov, mind, sl, sr, call, ret, clr, nop, end, ld, st, add, sub, addc, subc, addi, subi, and, or, andi, ori και jmp. Εάν το mnemonic ανήκει στο λεξικό, με τη μέθοδο «startswith» εξετάζονται οι τρείς περιπτώσεις, τοποθετώντας την κάθε λίστα σε μία νέα μεταβλητή με το αντίστοιχο όνομα. Με αυτή την ενέργεια, αν βρεθεί κάποια εντολή mnemonic, η οποία ανήκει σε κάποια από αυτές τις λίστες, επιστρέφει 'True' στην εκάστοτε κατηγορία. Η πρώτη λίστα προς έλεγχο είναι η «pref1», η οποία περιέχει εντολές που έχουν function bit ίσο με ένα. Η μεταβλητή που περιέχει τη λίστα προς έλεγχο ονομάζεται «fb1». Με ανάλογο τρόπο, η λίστα «pref0» περιέχει εντολές που έχουν μηδενικό function bit και η μεταβλητή ονομάζεται «fb0». Έπειτα δηλώνονται δύο μεταβλητές που ονομάζονται FB0, FB1 και αντιστοιχούν ανάλογα οι τιμές 0, 1. Στην αρχή του κώδικα, αρχικοποιήθηκε μια κενή λίστα που ονομάζεται «fbit» για να αποθηκεύεται το function bit. Τέλος, με τον προηγούμενο έλεγχο που έγινε με τη χρήση της «startswith», όταν η εντολή mnemonic ανήκει στην πρώτη κατηγορία τοποθετείται '1' στη λίστα. Εάν ανήκει στη δεύτερη κατηγορία στη λίστα τοποθετείται η τιμή '0'. Ακολουθεί ο κώδικας για την εύρεση του function bit.

```
for mnem in list5:
    pref1 = ['cmp', 'beq', 'bgt']
    pref0 = ['movi', 'mov', 'mind', 'sl', 'sr', 'call', 'ret', 'clr', 'nop', 'end',  'addc', 'subc'
             'ld', 'st', 'add', 'sub', 'addi', 'subi', 'and', 'or', 'andi', 'ori', 'jmp']
    if mnem in mydict:
        fb1 = mnem.startswith(tuple(pref1))
        fb0 = mnem.startswith(tuple(pref0))
        FB0 = 0
        FB1 = 1
        if fb1 == True:
          fbit.append(FB1)
        elif fb0 == True:
          fbit.append(FB0)
```

Δημιουργία και αποθήκευση δυαδικού αρχείου

Τα αποτελέσματα του συμβολομεταφραστή αποθηκεύονται σε ένα νέο αρχείο κειμένου με όνομα “outt.txt”. Η εισαγωγή των εντολών στο δυαδικό αρχείο γίνεται με τη σειρά που εμφανίζονται στην Εικόνα 1.3.1. Μέσα από ένα βρόγχο for διατρέχεται ανά γραμμή, κάθε λίστα που έχει δημιουργηθεί και περιέχει δυαδικό περιεχόμενο. Έπειτα, συναρμολογούνται τα δυαδικά μέρη της κάθε εντολής με τη χρήση της μεθόδου «zip» και τοποθετούνται στη σειρά όλα τα τμήματα της κάθε εντολής.

Το κάθε τμήμα της εντολής βρίσκεται σε αντίστοιχη λίστα. Αρχικά, εισάγεται το opcode από τη λίστα «list4» και αμέσως μετά το πρώτο operand από τη λίστα «listop1». Το μεσαίο στοιχείο της εντολής είναι το shamt, του οποίου η λίστα είναι η «listop3». Ακολουθεί το δεύτερο operand από τη λίστα «listop2» και τέλος, εισάγεται το function bit από την αντίστοιχη λίστα «fbit».

```
Output = open("outt.txt", "w")
   sys.stdout = Output
   for opc, op1, shmt, op2, func_bit in zip(list4, listop1, listop3, listop2, fbit):
      print(opc, op1, shmt, op2, func_bit)
   Output.close()
```

Το αρχείο “outt.txt” κατά την εμφάνιση των περιεχομένων ανάμεσα περιείχε κενό, ως διαχωριστικό στοιχείο. Το ακριβές περιεχόμενό του, αντιγράφεται σε ένα νέο αρχείο “output.txt”, στο οποίο με τη μέθοδο «replace» αντικαταστάθηκε το κενό διάστημα, εμφανίζοντας συνεχόμενα τα δυαδικά περιεχόμενα των 20 bit.

Κλείνοντας το πρόγραμμα και έχοντας τελειώσει όλες τις μετατροπές και ενέργειες, καλείται η συνάρτηση που περιέχει τον συμβολομεταφραστή. Αυτή η ενέργεια εκτελείται μέσω της εντολής ‘if __name__ == '__main__':’. Η ‘__name__’ είναι μία μεταβλητή, η οποία χρησιμοποιείται στην περίπτωση που ένα αρχείο χρειάζεται να εισαχθεί σε άλλο αρχείο, όπως αυτός ο συμβολομεταφραστής στο γραφικό περιβάλλον. Έτσι τελειώνει ο κώδικας του συμβολομεταφραστή(assembler).

3.3 Γραφικό περιβάλλον

Για να ολοκληρωθεί η παρούσα πτυχιακή εργασία, δημιουργήθηκε ένα νέο γραφικό περιβάλλον με τη χρήση της γλώσσας Python. Η συγκεκριμένη ενέργεια έγινε, έτσι ώστε το περιβάλλον του συμβολομεταφραστή να είναι εύχρηστο για τον χρήστη, βασισμένο στις ανάγκες που απαιτούν οι διεργασίες του. Παρακάτω αναλύεται το framework που χρησιμοποιείται, η δομή και οι λειτουργίες που εμφανίζονται καθώς και εικόνες με τα βασικά μενού.

3.3.1 Επιλογή framework για το γραφικό περιβάλλον (GUI)

Η δημιουργία του γραφικού περιβάλλοντος έγινε με τη χρήση του tkinter. Η συγκεκριμένη επιλογή έγινε διότι, είναι το μόνο framework που είναι ενσωματωμένο στην βασική βιβλιοθήκη της Python. Ο tkinter έχει πολλά πλεονεκτήματα, ένα από αυτά είναι η απλή σύνταξη των εντολών του. Επίσης, είναι μία πλατφόρμα cross-platform, πράγμα που σημαίνει πως μπορεί να χρησιμοποιηθεί σε διαφορετικούς τύπους υπολογιστών ή σε διαφορετικά πακέτα λογισμικού. Έτσι ο ίδιος κώδικας, λειτουργεί σε Windows, macOS και Linux. Τα οπτικά στοιχεία αποδίδονται με χρήση εγγενών στοιχείων του λειτουργικού συστήματος, επομένως η δημιουργία των εφαρμογών με τον tkinter, μοιάζουν να ανήκουν στην πλατφόρμα όπου εκτελούνται. Ωστόσο, ο tkinter είναι σχετικά ελαφρύτερο πλαίσιο σε σύγκριση με άλλα πλαίσια (framework).

3.3.2 Εισαγωγή λειτουργικής μονάδας και βιβλιοθήκης για το γραφικό περιβάλλον

Για τη δημιουργία του γραφικού περιβάλλοντος είναι απαραίτητη η δήλωση μίας λειτουργικής μονάδας(module) και η προσθήκη κάποιων σημείων κώδικα από άλλα αρχεία. Για να πραγματοποιηθεί η δήλωση της λειτουργικής μονάδας «os», πριν από το όνομα της γράφεται η εντολή «import», όπως ακριβώς γίνεται και η εισαγωγή μιας βιβλιοθήκης. Η «os» περιέχει δύο υποενότητες, την ‘os.sys’ και την ‘os.path’. Οι κατηγορίες αυτές είναι αφιερωμένες στο σύστημα και στους καταλόγους αντίστοιχα. Έτσι επιτρέπεται η χρήση σε λειτουργίες, που εξαρτώνται από το λειτουργικό σύστημα και αλληλεπιδρούν με το υποκείμενο λειτουργικό σύστημα με πολλούς

διαφορετικούς τρόπους. Για παράδειγμα, υπάρχει η δυνατότητα εργασίας σε αρχεία, γίνεται αλλαγή των μεταβλητών του περιβάλλοντος καθώς και μετακίνηση αρχείων.

Αμέσως μετά, γίνεται η δήλωση της βιβλιοθήκης «tkinter», η οποία είναι η τυπική βιβλιοθήκη για το γραφικό περιβάλλον (GUI) της Python. Όταν η Python συνδυάζεται με το tkinter παρέχει μια ισχυρή αντικειμενοστραφή διεπαφή, μέσο της εργαλειοθήκης tk. Επίσης, είναι μια εύκολη εργαλειοθήκη στη χρήση, για κάποιον αρχάριο προγραμματιστή, ο οποίος θέλει να δημιουργήσει μια εφαρμογή σε γραφικό περιβάλλον χρησιμοποιώντας το tkinter.

3.3.3 Προσθήκη από βιβλιοθήκες και άλλα αρχεία

Η δημιουργία του γραφικού περιβάλλοντος, απαιτεί την προσθήκη εργαλείων από την βιβλιοθήκη tkinter αλλά και την εισαγωγή συναρτήσεων από άλλα αρχεία. Αρχικά, με την δήλωση «from tkinter.messagebox import*», εισάγεται από τη βιβλιοθήκη «tkinter» το message box. Το message box είναι μια ενότητα που χρησιμοποιείται για την εμφάνιση μηνυμάτων στην εφαρμογή. Αυτή η ενότητα, παρέχει μια σειρά από λειτουργίες που χρησιμοποιούνται για την εμφάνιση ενός μηνύματος. Η επόμενη δήλωση είναι η «from tkinter.filedialog import*». Η προσθήκη του filedialog είναι ένα νέο παράθυρο, το οποίο διατίθεται σε διάφορους τύπους. Για παράδειγμα η εμφάνιση ενός παραθύρου, που ζητά την επιλογή ενός αρχείου και επιστρέφει το όνομα του συγκεκριμένου αρχείου επιλογής.

Στη συνέχεια, ακολουθούν δηλώσεις συναρτήσεων από άλλα αρχεία της Python, τα οποία συμβάλουν στην ολοκληρωμένη εφαρμογή του συμβολομεταφραστή. Αρχικά, με την δήλωση «from Assembler import assembler», γίνεται η εισαγωγή της συνάρτησης «assembler» από το αρχείο του συμβολομεταφραστή, η οποία εμπεριέχει όλο τον κώδικα που τον απαρτίζει. Με ανάλογο τρόπο πραγματοποιείται και η εισαγωγή των δύο επόμενων συναρτήσεων από δύο διαφορετικά αρχεία. Το πρώτο αρχείο είναι αυτό, στο οποίο εμφανίζονται τα σφάλματα που μπορεί να προκύψουν και η εισαγωγή του γίνεται με την εντολή «from Errors import errors». Το δεύτερο αρχείο είναι αυτό, που περιέχει τα αποτελέσματα εξόδου του συμβολομεταφραστή σε δυαδική μορφή. Η εισαγωγή πραγματοποιείται με τη δήλωση «from Output import output». Έτσι, τελειώνει το τμήμα με των δηλώσεων από τις βιβλιοθήκες και τις προσθήκες συναρτήσεων από άλλα αρχεία που θα χρησιμοποιηθούν παρακάτω.

3.3.4 Κλάση και αντικείμενα

Για τη δημιουργία του γραφικού περιβάλλοντος χρησιμοποιήθηκε μία κλάση. Η Python είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού. Σχεδόν τα πάντα σε αυτή είναι ένα αντικείμενο, με τις ιδιότητες και τις μεθόδους του. Η κλάση είναι σαν ένας κατασκευαστής αντικειμένων ή ένα "σχεδιάγραμμα" για τη δημιουργία αντικειμένων. Για τη δήλωση μιας κλάσης, χρησιμοποιείται η λέξη κλειδί «class» κενό και το όνομα της, κι έπειτα ακολουθεί ο υπόλοιπος κώδικας.

Για την κατανόηση της έννοιας "κλάση" και "αντικείμενο" είναι απαραίτητη η κατανόηση και χρήση της συνάρτησης «__init__()». Όλες οι κλάσεις έχουν μια συνάρτηση που ονομάζεται έτσι, η οποία εκτελείται πάντα όταν ξεκινάει η κλάση. Η λειτουργία της είναι να αντιστοιχεί τιμές και ιδιότητες αντικειμένων ή άλλες λειτουργίες που είναι απαραίτητες να γίνουν, κατά την δημιουργία του αντικειμένου. Επίσης, τα αντικείμενα μπορούν να περιέχουν μεθόδους. Οι μέθοδοι σε αντικείμενα είναι συναρτήσεις που ανήκουν στο αντικείμενο.

Η παράμετρός «self», χρησιμοποιείται για την πρόσβαση σε μεταβλητές που ανήκουν στην κλάση. Η ονομασία «self» μπορεί να αντικατασταθεί από οποιοδήποτε όνομα αρκεί να είναι η πρώτη παράμετρος οποιασδήποτε λειτουργίας στην κλάση.

3.3.5 Επεξήγηση του κώδικα για την δημιουργία του γραφικού περιβάλλοντος (GUI)

Σε αυτή την ενότητα γίνεται η επεξήγηση του κώδικα, με τον οποίο υλοποιήθηκε το γραφικό περιβάλλον με σκοπό την ευκολότερη χρήση του συμβολομεταφραστή (assembler). Στο γραφικό περιβάλλον απεικονίζεται ένα αρχείο κειμένου(Text Editor). Τα περιεχόμενα του παραθύρου είναι ένας κενός χώρος, στον οποίο γίνεται η συγγραφή των εντολών(text area). Επίσης, περιέχει ένα βασικό μενού, που μέσα σε αυτό βρίσκονται οι κύριες λειτουργίες που είναι χρήσιμες για τον χρήστη.

Αρχικά, η ανάπτυξη κώδικα ξεκινά με την εισαγωγή της απαραίτητης βιβλιοθήκης «tkinter», από την οποία γίνεται η προσθήκη του message box και του filedialog. Επίσης, εισάγονται από άλλα τρία αρχεία μέσω συναρτήσεων, τμήματα κώδικα χρήσιμα για τις λειτουργίες του γραφικού περιβάλλοντος. Η πρώτη προσθήκη συνδέει το πρόγραμμα του συμβολομεταφραστή με το γραφικό περιβάλλον, μέσω της

συνάρτησης assembler. Σε αυτή την περίπτωση καλείται όλο το πρόγραμμα, εφόσον είναι εμφωλευμένο μέσα σε μία συνάρτηση. Έπειτα, καλούνται άλλα δύο αρχεία που ως συναρτήσεις, τα οποία περιέχουν το αρχείο εξόδου και το αρχείο σφαλμάτων του συμβολομεταφραστή. Παρακάτω, αναφέρονται οι βασικές δηλώσεις που χρειάστηκαν.

```
import os

import tkinter

from tkinter import *

from tkinter.messagebox import *

from tkinter.filedialog import *

from Assembler import assembler

from Errors import errors

from Output import output
```

Αμέσως μετά από την εισαγωγή και προσθήκη βασικών στοιχείων πραγματοποιείται η δημιουργία μιας κλάσης με το όνομα Notepad. Το πρόγραμμα είναι εμφωλευμένο μέσα σε μία κλάση. Ξεκινώντας ορίζονται τα κύρια χαρακτηριστικά του παραθύρου. Αρχικά, ορίζεται η κενή περιοχή κειμένου(text area) και η περιοχή για τη δημιουργία του μενού. Στη συνέχεια, γίνεται η δήλωση για τα βασικά μενού τα οποία είναι τα εξής: το μενού «File», «Edit», «CallProg2» και το «HelpMenu». Μια ακόμη προσθήκη είναι το Scrollbar ή αλλιώς μπάρα κύλισης, στο οποίο γίνεται η κύλιση κειμένου προς μία προκαθορισμένη κατεύθυνση.

Όπως όλες οι κλάσεις έτσι και σε αυτή, υπάρχει μία συνάρτηση με το όνομα «__inti__», η οποία εκτελείται πάντα όταν ξεκινά η κλάση. Η χρησιμότητα της είναι η αντιστοιχία τιμών σε ιδιότητες αντικειμένου ή σε άλλες λειτουργίες, οι οποίες είναι απαραίτητες για τη λειτουργία του αντικειμένου. Μέσα στη συνάρτηση η πρώτη δήλωση είναι το εικονίδιο, που έχει το παράθυρο(πάνω αριστερά), το οποίο μοιάζει σε σημειωματάριο. Έπειτα, ακολουθεί η δήλωση του αρχικού τίτλου, διότι ο χρήστης μπορεί να δώσει στο αρχείο όποιο όνομα θέλει. Το προκαθορισμένο όνομα του τίτλου είναι «Untitled – My Assembler». Ακόμη ένα βασικό χαρακτηριστικό είναι το

μέγεθος του παραθύρου, το οποίο ορίζεται με 600x400 pixel. Το μέγεθος μπορεί να τροποποιηθεί από το χρήστη, όσο το παράθυρο είναι ανοιχτό χειροκίνητα μεγαλώνοντας ή μικραίνοντάς το. Ακολουθεί, η αυτόματη προσαρμογή μεγέθους για το χώρου του κειμένου, οπού χρησιμοποιείται σαν κειμενογράφος για τη συγγραφή των εντολών. Το «status bar» ορίζεται στο κάτω μέρος από το χώρου κειμένου, καθώς εκεί δίνονται πληροφορίες για τον τρόπο εμφάνισης των μηνυμάτων, όπως η γραμματοσειρά, το μέγεθος και το σημείο γραφής σε αυτό το χώρο . Σε αυτό το χώρο εμφανίζονται μηνύματα σχετικά με τις ενέργειες που πράττει ο χρήστης.

3.3.6 Δηλώσεις των μενού του αρχείου κειμένου

Αρχικά, πραγματοποιούνται οι δηλώσεις για τις βασικές λειτουργίες των μενού.

Μενού File

Το πρώτο μενού έχει το όνομα «File» και περιέχει τέσσερα υπομενού. Πρώτα γίνεται η δήλωση των υπομενού και τελευταία η δήλωση του κυρίως μενού. Το πρώτο υπομενού ονομάζεται «New», με τη χρήση του γίνεται η δημιουργία ενός νέου αρχείου. Το δεύτερο υπομενού είναι το «Open», το οποίο ανοίγει ένα νέο παράθυρο που οπτικά μοιάζει με το «Open», σε οποιαδήποτε άλλη εφαρμογή του λειτουργικού συστήματος, που έχει ο χρηστής στον υπολογιστή του. Στο παράθυρο που ανοίγει όταν χρησιμοποιεί ο χρήστης αυτό το υπομενού, μπορεί να επιλέξει ένα αρχείο για να το ανοίξει, μέσω του γραφικού περιβάλλοντος του αρχείου κειμένου. Το τρίτο υπομενού είναι το «Save», το οποίο αποθηκεύει το αρχείο που είναι ήδη ανοιχτό στο παράθυρο. Με το υπομενού «Exit», ο χρήστης δηλώνει την έξοδο από το παραθυρικό περιβάλλον. Το «Exit» διαχωρίζεται από τα άλλα τρία υπομενού, με μια οριζόντια μαύρη γραμμή. Παρατίθενται οι δηλώσεις για το μενού «File».

```
self.thisFileMenu.add_command(label="New", command=self.newFile)

self.thisFileMenu.add_command(label="Open", command=self.openFile)

self.thisFileMenu.add_command(label="Save", command=self.saveFile)

self.thisFileMenu.add_separator()

self.thisFileMenu.add_command(label="Exit", command=self.quitApplication)

self.thisMenuBar.add_cascade(label="File", menu=self.thisFileMenu)
```

Για την δήλωση του υπομενού, η πρώτη εντολή αρχίζει με την παράμετρο «self». Η παράμετρος αυτή συνδέει το μενού με τη συνάρτηση, που περιέχει τον κώδικα για τη λειτουργία του. Ακολουθεί, το όνομα στο οποίο ανήκει το μενού που η δήλωση του είχε γίνει στην αρχή της κλάσης(σελίδα 97 στο παράρτημα). Με τη χρήση της μεθόδου «add_command» μέσα σε παρενθέσεις, προστίθεται ο τίτλος του υπομενού γράφοντας label, ίσον και το επιθυμητό όνομα για αυτό. Επίσης, γίνεται κλήση της συνάρτησης, που περιέχει τη λειτουργία του υπομενού με την εντολή command, ίσον και το όνομα της.

Για τον διαχωρισμό των υπομενού μέσω μίας μαύρης οριζόντιας γραμμής, γράφεται πρώτα η παράμετρος «self», ακολουθεί το όνομα του μενού που έχει δοθεί κατά την δήλωση του και τελευταία γράφεται η μέθοδος «add_seperator», η οποία προσθέτει την διαχωριστική γραμμή.

Στη δημιουργία του γονικού μενού, υπάρχουν κάποιες διαφοροποιήσεις. Πρώτη γράφεται η παράμετρος «self», ακολουθεί η δήλωση του κυρίως μενού με το όνομα δήλωσης του στην αρχή της κλάσης. Στην συνέχεια, με τη χρήση της μεθόδου «add_cascade», δημιουργείται ένα νέο ιεραρχικό μενού συσχετίζοντας δεδομένο υπομενού με ένα γονικό. Μέσα σε παρενθέσεις μετά τη μέθοδο, γίνεται η δήλωση του τίτλου και του μενού στη αρχική δήλωση.

Με ανάλογο τρόπο γίνεται η κατασκευή και των υπόλοιπων μενού που ακολουθούν..

Μενού Edit

Το δεύτερο μενού ονομάζεται «Edit», όπως λέει και το όνομα του αφορά την επεξεργασία του κειμένου και περιέχει τρία υπομενού. Το πρώτο είναι το «Cut», που κάνει αποκοπή του επιλεγμένου κειμένου. Δεύτερο είναι το «Copy», με το οποίο γίνεται η αντιγραφή του κειμένου που έχει επιλεχθεί. Το τρίτο υπομενού ονομάζεται «Paste», το οποίο κάνει επικόλληση το κείμενο που έχει αντιγραφεί.

Μενού Make it Binary

Το τρίτο μενού ονομάζεται «Make it Binary» και σχετίζεται με την επεξεργασία του κειμένου με τη χρήση του συμβολομεταφραστή. Αυτό το μενού έχει τρία υπομενού. Το πρώτο ονομάζεται «Convert» και καλεί τη συνάρτηση «assembler» από το πρόγραμμα του συμβολομεταφραστή(Assembler), για την μετατροπή του τρέχοντος

αρχείου σε δυαδική μορφή. Το δεύτερο υπομενού είναι το «Output», με το οποίο καλείται η συνάρτηση «output» από το αρχείο εξόδου, που περιέχει το αποτέλεσμα του μενού «Convert». Το τρίτο και τελευταίο υπομενού ονομάζεται «Error Log» και καλώντας τη συνάρτηση «errors», από το αρχείο εξόδου, ανοίγει το αρχείο σφαλμάτων.

Μενού Help

Το τελευταίο μενού είναι το «Help», το οποίο περιέχει μόνο ένα υπομενού, το «About Notebook». Με τη χρήση του εμφανίζεται ένα μήνυμα για τον συμβολομεταφραστή, σε ένα παράθυρο πληροφοριών (showinfo).

Ανάλογα με το μέγεθος του κειμένου διαμορφώνεται αυτόματα και το μέγεθος του scrollbar με τη χρήση των δύο επόμενων εντολών.

```
self.thisScrollBar.config(command=self.thisTextArea.yview)
self.thisTextArea.config(yscrollcommand=self.thisScrollBar.set)
```

Οι δηλώσεις των μενού βρίσκονται στο παράρτημα στις σελίδες 97 και 98.

3.3.7 Δημιουργία συναρτήσεων για τη λειτουργία των μενού

Εφόσον έχουν οριστεί τα μενού, αμέσως μετά γίνεται η δήλωση των συναρτήσεων, μέσα από τις οποίες καθορίζονται οι λειτουργίες των υπομενού. Οι συναρτήσεις ορίζονται με τη σειρά που έχουν αναφερθεί κατά τη δημιουργία τους.

Η πρώτη συνάρτηση αφορά το μενού «New». Η δήλωσή της γίνεται με την χρήση της εντολής «def», ακολουθεί το όνομα «newFile» και μέσα στις παρενθέσεις υπάρχει η παράμετρος «self» που συνδέει τη συνάρτηση με την κλάση. Με τη χρήση του υπομενού «New», δημιουργείται ένα νέο αρχείο με τίτλο Untitled. Για τη δημιουργία του νέου αρχείου, διαγράφονται από την περιοχή κειμένου οι χαρακτήρες που υπάρχουν από τον πρώτο έως τον τελευταίο, με την εντολή «delete(1.0, END)». Τέλος ενημερώνεται το status, εμφανίζοντας το μήνυμα "New file Created".

Η δεύτερη συνάρτηση αφορά το μενού «Open». Η δήλωση της πραγματοποιείται με τον ίδιο τρόπο της «New». Αρχικά, ανοίγει ένα παράθυρο, στο οποίο εμφανίζονται τα αρχεία που υπάρχουν στον εκάστοτε επιλεγμένο φάκελο. Στη συνέχεια, ο χρήστης μπορεί να επιλέξει αρχείο οποιουδήποτε τύπου, κι όχι μόνο αρχεία κειμένου με την

κατάληξη ".txt". Το αρχείο επιλογής, οποιουδήποτε τύπου κι αν είναι, κατά το άνοιγμα του αλλάζει ο τύπος του και μετατρέπεται σε αρχείου κειμένου με κατάληξη '.txt'. Εφόσον γίνει η επιλογή κάποιου αρχείου, εμφανίζονται τα περιεχόμενα του στην περιοχή κειμένου. Το αρχείο που επιλέγει ο χρήστης, για να γίνει αρχείο εισόδου στον συμβολομεταφραστή, ήταν μία περίπτωση που χρειάστηκε μια διεργασία καθώς αποτελούσε ένα προγραμματιστικό πρόβλημα. Για τις ανάγκες σύνδεσης του γραφικού περιβάλλοντος με τον συμβολομεταφραστή, δημιουργείται ένα νέο αρχείο με όνομα 'temp.txt', στο οποίο αντιγράφεται το περιεχόμενο του αρχείου επιλογής από το χρήστη. Το μεταβατικό αρχείο 'temp', φιλοξενεί το περιεχόμενο οποιουδήποτε αρχείου επιλεγεί. Έτσι, με αυτή τη μετατροπή ο συμβολομεταφραστής διαβάζει ως είσοδο ένα αρχείο, που πάντα έχει το ίδιο όνομα αλλά με διαφορετικό περιεχόμενο. Το αρχείο 'temp' εμφανίζεται κατά τη δημιουργία του στον ίδιο φάκελο με το αρχείο εισόδου, αλλά δεν απασχολεί το χρήστη. Έτσι, με αυτό τον αρχείο έγινε η επίλυση του προβλήματος. Τέλος, ενημερώνεται ο τίτλος του αρχείου και στο status bar εμφανίζεται το μήνυμα "Opened Successfully".

Η τρίτη συνάρτηση ονομάζεται «Save», αλλά λειτουργεί και σαν 'Save as', καθώς υπάρχουν δύο περιπτώσεις αποθήκευσης. Στην πρώτη περίπτωση, όταν το αρχείο έχει δημιουργηθεί από την αρχή μέσω του μενού «New», για την αποθήκευσή του, ανοίγει ένα παράθυρο, το οποίο μοιάζει με αυτό που ανοίγει για τη λειτουργία του 'save as', στο εκάστοτε λειτουργικό σύστημα. Καθώς ανοίγει το νέο παράθυρο, έχει ως προεπιλεγμένο όνομα αποθήκευσης το 'Untitled.txt'. Ο χρήστης δίνει το όνομα που επιθυμεί και πατώντας "Save" αυτόματα, αλλάζει ο τίτλος του παραθύρου σε αυτόν που δόθηκε. Επίσης, σε αυτή την περίπτωση ο χρήστης μπορεί να αλλάξει και τον προεπιλεγμένο φάκελο αποθήκευσης. Η δεύτερη περίπτωση είναι όταν το αρχείο προϋπάρχει και ο χρήστης ανοίγει το αρχείο μέσα από την επιλογή «Open». Έτσι, όταν κάνει 'Save', το αρχείο αποθηκεύεται με το όνομα, που ήδη έχει στην ίδια τοποθεσία. Τέλος, το status εμφανίζει το μήνυμα "Saved Successfully".

Η τελευταία συνάρτηση του μενού 'File' είναι το «Exit», το οποίο κλείνει το παράθυρο με την χρήση της εντολής destroy.

```
def quitApplication(self):
self.root.destroy()
```

Το μενού «Edit» έχει τρία υπομενού, άρα και τρείς συναρτήσεις συνολικά, μία για κάθε υπομενού. Η πρώτη συνάρτηση αφορά το μενού «Cut». Ξεκινώντας πραγματοποιείται ο ορισμός της εντολής def, το όνομα είναι ίδιο με τη λειτουργία της συνάρτησης, δηλαδή την cut και τέλος, μέσα σε εισαγωγικά βρίσκεται η παράμετρος self. Στη συνέχεια, γίνεται χρήση της μεθόδου «event_generate». Για την κατανόηση της, χρειάζεται να αναφερθεί πως το event είναι μία ενέργεια που συμβαίνει στην εφαρμογή και μετά από αυτή, η εφαρμογή πρέπει να αντιδράσει. Για παράδειγμα, μία τέτοια ενέργεια μπορεί να είναι όταν ο χρήστης πατά ένα πλήκτρο ή κάνει κλικ, ή σύρει το ποντίκι του. Η μέθοδος «event_generate» προκαλεί ένα συμβάν, το οποίο ενεργοποιείται από μια ενέργεια του χρήστη σαν αυτές που προαναφέρθηκαν. Το όρισμα της μεθόδου, περιγράφει το συμβάν που ενεργοποιεί. Έτσι, το όρισμα «Cut» καθώς ο χρήστης επιλέγει κάποιο σημείο του κειμένου και στη συνέχεια, πατήσει στο μενού «Cut» το κείμενο αποκόπτεται. Ακολουθεί η παρουσίαση του κώδικα της συνάρτησης.

```
def cut(self):
    self.thisTextArea.event_generate("<<Cut>>")
    self.status.set("Cut Successfullv")
```

Με τον ίδιο τρόπο δόμησης και μεθόδων, γίνεται η δημιουργία των άλλων δύο συναρτήσεων Copy και Paste, του μενού «Edit». Ο κώδικας παραθέτεται στις σελίδες 99 και 100 του παραρτήματος. Όταν χρησιμοποιηθεί κάποιο από τα τρία παραπάνω υπομενού, ενημερώνεται το status με ανάλογο μήνυμα. Για το υπομενού «Cut» εμφανίζεται το μήνυμα «Cutted Successfully», για το «Copy» εμφανίζεται «Copied Successfully» και τέλος για το «Paste» εμφανίζεται «Pasted Successfully».

Στο μενού «Make it binary» καλούνται συναρτήσεις, οι οποίες έχουν γραφεί σε εξωτερικά προγράμματα. Το μενού «Convert» καλεί τη συνάρτηση “assembler” από το πρόγραμμα ‘Assembler’ του συμβολομεταφραστή, που κάνει την μετατροπή του αρχείου σε δυαδική μορφή. Το μενού «output», καλεί τη συνάρτηση “output” από το πρόγραμμα ‘Output’. Σε αυτό το αρχείο εμφανίζεται το αρχείο εξόδου, που παράγεται από το συμβολομεταφραστή και έχει μετατραπεί σε δυαδική μορφή. Τέλος, το μενού «Error Log» καλεί τη συνάρτηση “errors” από το πρόγραμμα ‘Errors’, το οποίο περιέχει τα συντακτικά σφάλματα, μετά τη μετατροπή του αρχείου.

Το τελευταίο μενού ονομάζεται «Help» και περιέχει ένα υπομενού, το οποίο ονομάζεται “About Assembler”. Σε αυτό ορίζεται η συνάρτηση με την εντολή def, ακολουθεί το όνομα της συνάρτησης ‘showAbout’ και μέσα σε παρένθεση υπάρχει η παράμετρος self. Με τη χρήση της συνάρτησης «showinfo» και του περιεχομένου της, που είναι ένα σχετικό μήνυμα για τον χρήστη, εμφανίζεται ένα νέο παράθυρο ενημέρωσης που εμφανίζει το μήνυμα. Τέλος ενημερώνεται το status και εμφανίζει το μήνυμα “Opened About”.

Κλείνοντας το πρόγραμμα του αρχείου κειμένου, με τη δημιουργία μιας συνάρτησης ‘run’ τρέχει η κύρια εφαρμογή. Τέλος, δίνεται το όνομα notepad σε μια μεταβλητή, η οποία περιέχει το ολοκληρωμένο όνομα της κλάσης. Με τον τρόπο αυτό, γράφοντας τη μεταβλητή, τελεία και τέλος τη συνάρτηση ‘run’ τρέχει το κυρίως πρόγραμμα.

3.3.8 Αρχείο Εξόδου

Το αρχείο εξόδου είναι ένα αρχείο κειμένου, που παράγεται από το πρόγραμμα του συμβολομεταφραστή. Για την διευκόλυνση του χρήση, το αρχείο αυτό μπορεί να το ανοίξει μέσα από το μενού του αρχείου κειμένου(Text Editor).

Το αρχείο κειμένου ονομάζεται «Output». Το πρόγραμμα είναι εμφωλευμένο μέσα σε μία συνάρτηση με το όνομα «output». Ο λόγος της δημιουργίας της παρούσας συνάρτησης , είναι για να μπορεί το πρόγραμμα του Text Editor να την καλέσει μέσα από το μενού «Make it Binary», επιλέγοντας το υπομενού «Output».

Αρχικά, εισάγεται η βιβλιοθήκη tkinter. Στη συνέχεια, ορίζονται τα βασικά χαρακτηριστικά του παραθύρου όπως το μέγεθος, το οποίο είναι ίδιο με το αρχικό παράθυρο του Text Editor, δηλαδή 600x400 pixel. Έπειτα, ορίζεται ο τίτλος του παραθύρου «Output of Assembler» και το εικονίδιο που έχει επάνω αριστερά, το οποίο είναι ένα βελάκι που απεικονίζει την έξοδο. Μετά από τη δήλωση των βασικών χαρακτηριστικών, γράφεται η συνάρτηση ‘do open’, η οποία διαβάζει το αρχείο εξόδου «output.txt». Τα περιεχόμενα του, αποθηκεύονται σε μία μεταβλητή με το όνομα ‘content’. Για την εμφάνιση του με τη χρήση της συνάρτησης delete, σβήνεται ότι υπάρχει στο χώρο του κειμένου. Εφόσον η περιοχή είναι άδεια, με τη συνάρτηση insert, γίνεται η εμφάνιση του αρχείου, το οποίο περιέχει η μεταβλητή content. Όταν πατηθεί το κουμπί ‘open’ καλείται η συνάρτηση ‘do_open’, η οποία εισάγει το περιεχόμενο του «output.txt». Τέλος, δημιουργείται ένας χώρος κειμένου, όπου εκεί

τοποθετείται το κείμενο από το αρχείο εξόδου και κλείνει η αρχική συνάρτηση. Τέλος, με τη συνάρτηση «if __name__ == '__main__': », καλείται η αρχική συνάρτηση που τρέχει τον κώδικα που περιέχει. Στην ουσία είναι ένα απλό παράθυρο που περιλαμβάνει ένα κουμπί με το όνομα «open» για να ανοίγει το αρχείο εξόδου και ένα κενό χώρο που εμφανίζεται το κείμενο.

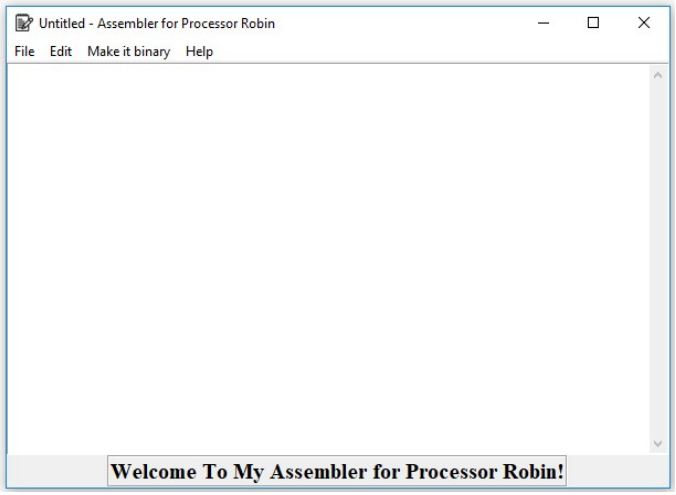
3.3.9 Αρχείο Σφαλμάτων

Η ανάγκη για αποθήκευση όλων των σφαλμάτων σε ένα αρχείο, δημιούργησε το συγκεκριμένο το αρχείο Python που ονομάζεται «Errors». Το αρχείο αυτό είναι ενσωματωμένο στο αρχείο του Text Editor, μέσα από το μενού «Make it Binary» και το υπομενού «Error Log».

Το αρχείο αυτό έχει την ίδια δομή με το αρχείο εξόδου. Το πρόγραμμα είναι εμφωλευμένο μέσα σε μία συνάρτηση με το όνομα «errors». Αρχικά, εισάγονται οι δύο βασικές βιβλιοθήκες tkinter και os(operating system), για τη δημιουργία του γραφικού περιβάλλοντος. Έπειτα, δημιουργείται το παράθυρο το μέγεθός του είναι ίδιο με τα άλλα δύο αρχεία, 600x400 pixel. Στην συνέχεια, ορίζεται ο τίτλος που έχει το παράθυρο: «Errors in your Assembler», καθώς επίσης και το εικονίδιο του παραθύρου, το οποίο απεικονίζει ένα κίτρινο τρίγωνο με ένα θαυμαστικό. Ακολούθως, γράφεται η συνάρτηση για τη λειτουργία του κουμπιού. Η συνάρτηση ονομάζεται «do_open». Το όνομα του αρχείου εξόδου είναι «Error output.txt» και τοποθετείται σε μία νέα μεταβλητή «x» για ευκολία κατά την κλήση του. Καθώς διαβάζεται το αρχείο με την εντολή open, πραγματοποιείται έλεγχος εάν το μέγεθος του είναι μηδενικό, που σημαίνει πως είναι κενό και δεν υπάρχει σφάλματα. Σε αυτή την περίπτωση εκτυπώνεται ένα μήνυμα, πως δεν υπάρχει κάποιο λάθος στον κώδικα. Αλλιώς με χρήση της συνάρτησης delete, σβήνεται ότι υπάρχει στο χώρο του κειμένου και με την εντολή insert, εμφανίζεται το περιεχόμενο του αρχείου. Στη συνέχεια, ορίζεται η λειτουργία του κουμπιού με το όνομα «open» και τη λειτουργία «do_open». Τέλος, δημιουργείται ο χώρος για το κείμενο, όπου εκεί εμφανίζονται τα λάθη από το αρχείο ‘txt’. Το πρόγραμμα με το αρχείο εξόδου και το αρχείο σφαλμάτων βρίσκεται στο παράρτημα στις σελίδες 101 και 102 .

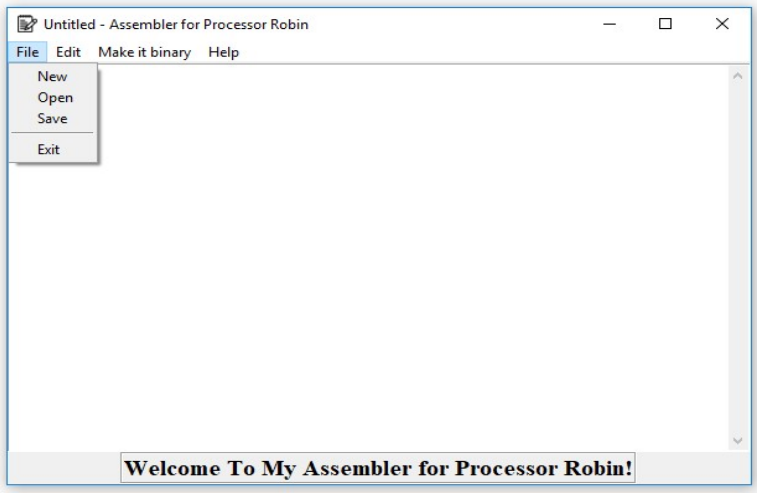
3.4 Παρουσίαση γραφικού περιβάλλοντος (GUI)

Στις παρακάτω εικόνες παρουσιάζεται το γραφικό περιβάλλον. Στην Εικόνα 3.4.1 απεικονίζεται το αρχικό παράθυρο του Text Editor όταν ανοίγει. Περιέχει στο χώρο του ένα βασικό μενού, το οποίο φαίνεται στο επάνω μέρος του. Στο κέντρο του παραθύρου βρίσκεται ο χώρος, στον οποίο ο χρήστης μπορεί να γράψει τις εντολές ή να ανοίξει κάποιο αρχείο και εκεί να προβληθεί., Στο κάτω μέρος του εμφανίζεται ένα μήνυμα 'καλωσορίσματος' στο status bar.



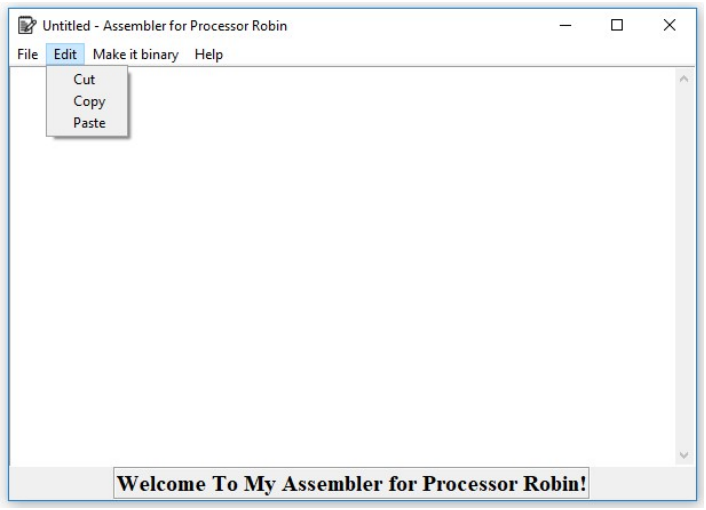
Εικόνα 3.4.1 - Αρχική όψη παραθύρου του Text Editor

Τα βασικά μενού είναι τέσσερα και το καθένα από αυτά έχει τουλάχιστον ένα υπομενού. Το πρώτο μενού στην Εικόνα 3.4.2, ονομάζεται «File». Περιέχει τέσσερα υπομενού(New,Open,Save,Exit), μέσα από τα οποία πραγματοποιούνται βασικές λειτουργίες του αρχείου.



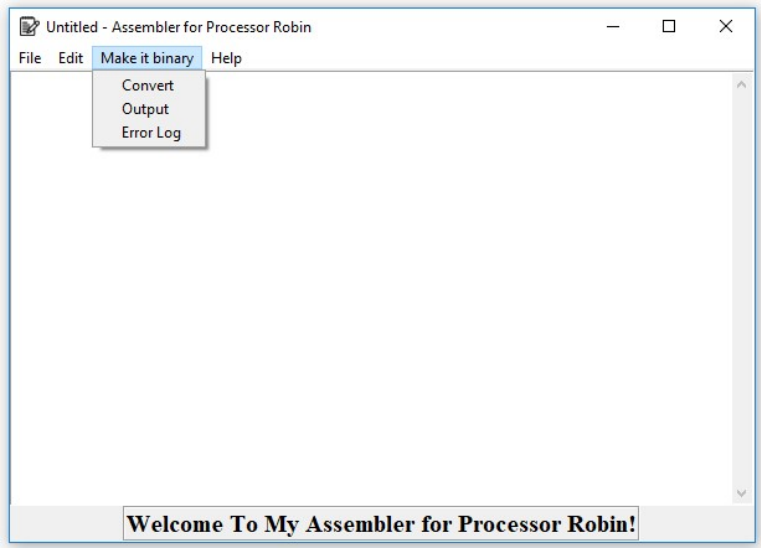
Εικόνα 3.4.2 - Εμφάνιση των υπομενού "File"

Το δεύτερο μενού στην Εικόνα 3.4.3, ονομάζεται «Edit». Περιέχει τρία υπομενού (Cut,Copy,Paste), που αφορούν την επεξεργασία του κειμένου. Το υπομενού Cut πραγματοποιεί την αποκοπή του επιλεγμένου κειμένου. Με την Copy αντιγράφεται το κείμενο και με το υπομενού Paste γίνεται η επικόλληση .



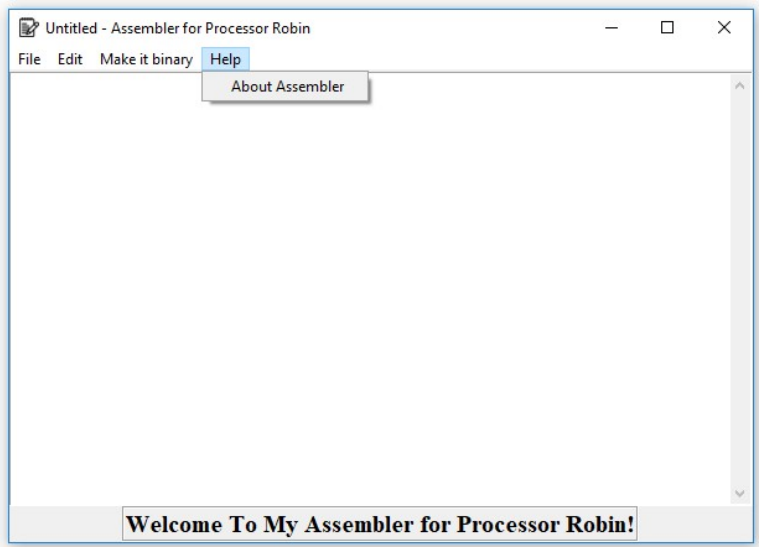
Εικόνα 3.4.3 – Εμφάνιση των υπομενού "Edit"

Το τρίτο μενού στην Εικόνα 3.4.4, ονομάζεται «Make it Binary». Περιέχει τρία υπομενού(Convert, Output, Error Log). Το υπομενού Convert μετατρέπει το αρχείο σε δυαδική μορφή. Το αμέσως επόμενο είναι το Output, το οποίο ανοίγει ένα νέο παράθυρο όπου εκεί εμφανίζεται το αρχείο εξόδου. Τελευταίο υπομενού είναι το Error Log, στο οποίο εμφανίζονται σε νέο παράθυρο τα σφάλματα του κώδικα.



Εικόνα 3.4.4 - Εμφάνιση των υπομενού του "Make it binary"

Το τελευταίο μενού ονομάζεται «Help», και περιέχει ένα υπομενού το About Assembler. Αυτό εμφανίζει ένα σχετικό μήνυμα σχετικό με τον συμβολομεταφραστή.



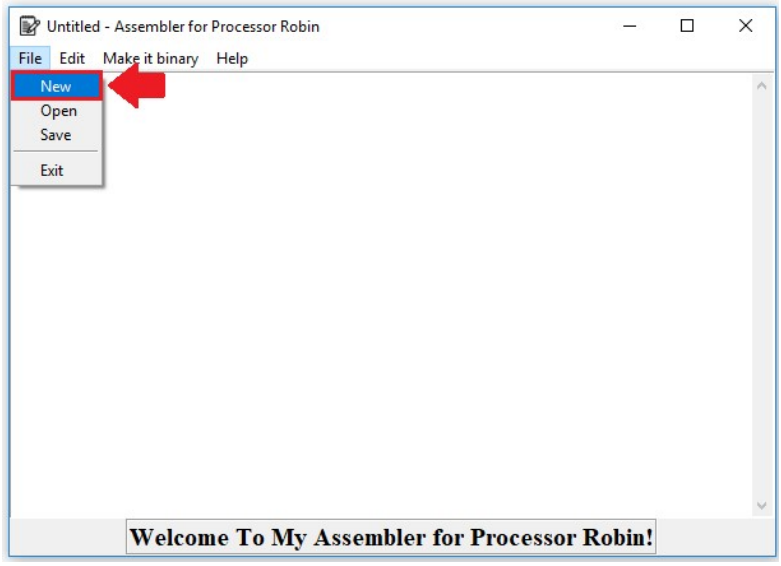
Εικόνα 3.4.5 - Εμφάνιση του υπομενού "Help"

Κεφάλαιο 4

Αποτελέσματα της εργασίας και Εγχειρίδιο χρήσης Text Editor

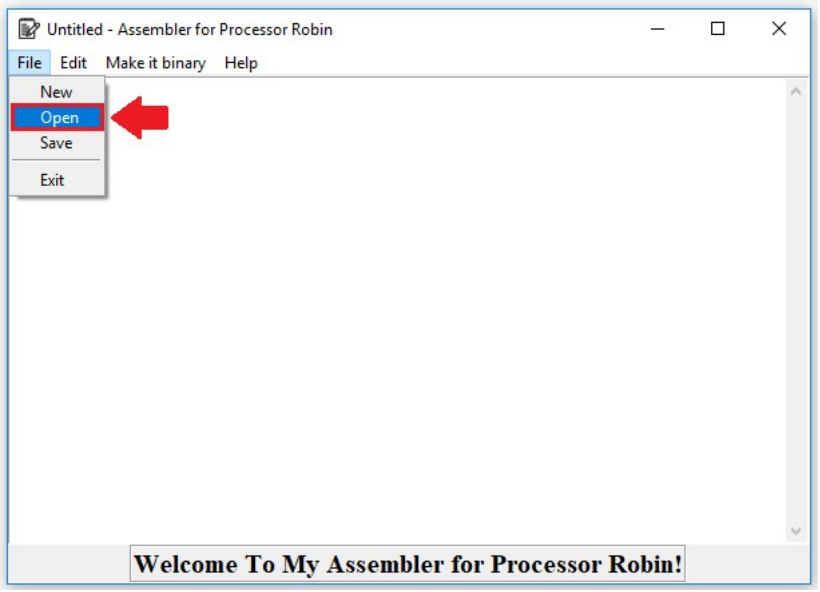
Σε αυτό το κεφάλαιο δίνονται βασικές οδηγίες για τον τρόπο χρήσης του Text Editor και παρουσιάζονται τα αποτελέσματα της εργασίας.

Ξεκινώντας από το αρχικό μενού «File», με το μενού «New» γίνεται η δημιουργία ενός νέου αρχείου κειμένου με προκαθορισμένο το όνομα 'Untitled'.



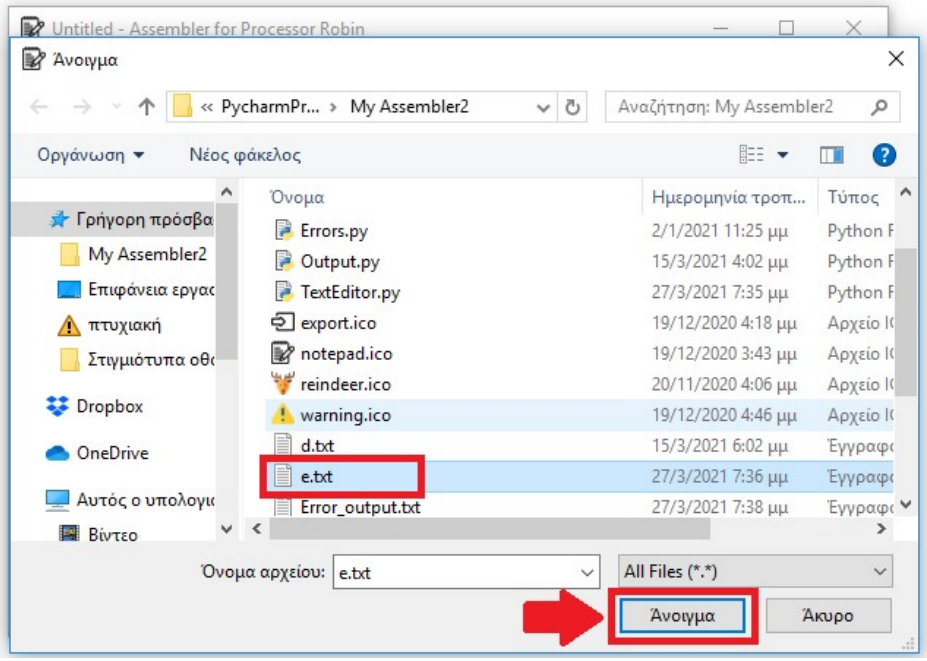
Εικόνα 4.1 - Υπομενού "New"

Το δεύτερο μενού είναι το «Open», το οποίο χρησιμοποιείται για να ανοίξει ένα αρχείο που ήδη υπάρχει.

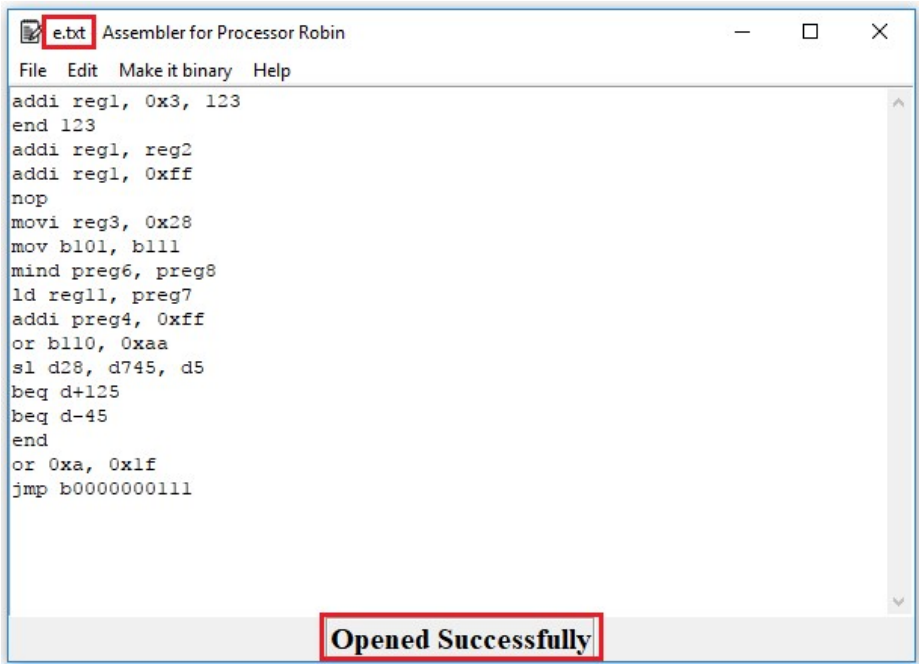


Εικόνα 4.2 - Υπομενού "Open"

Εφόσον χρησιμοποιηθεί το μενού «Open», ανοίγει ένα νέο παράθυρο για να επιλέξει ο χρήστης ένα αρχείου. Μόλις επιλεχθεί πατάει το κουμπί «Άνοιγμα». Έτσι εμφανίζεται το περιεχόμενο του επιλεγμένου αρχείου στο χώρο του κειμένου του Text Editor.

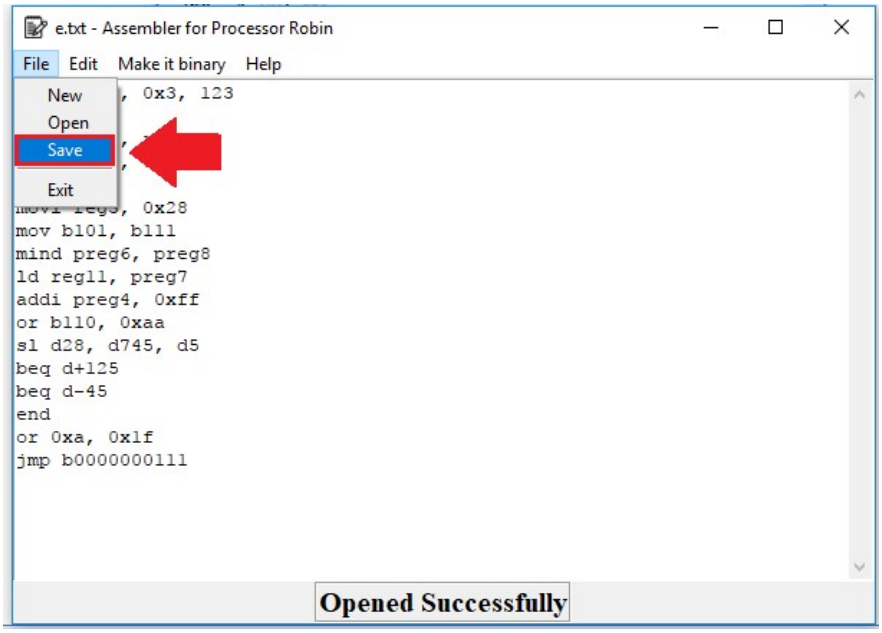


Εικόνα 4.3 – Επιλογή αρχείου από φάκελο και άνοιγμα



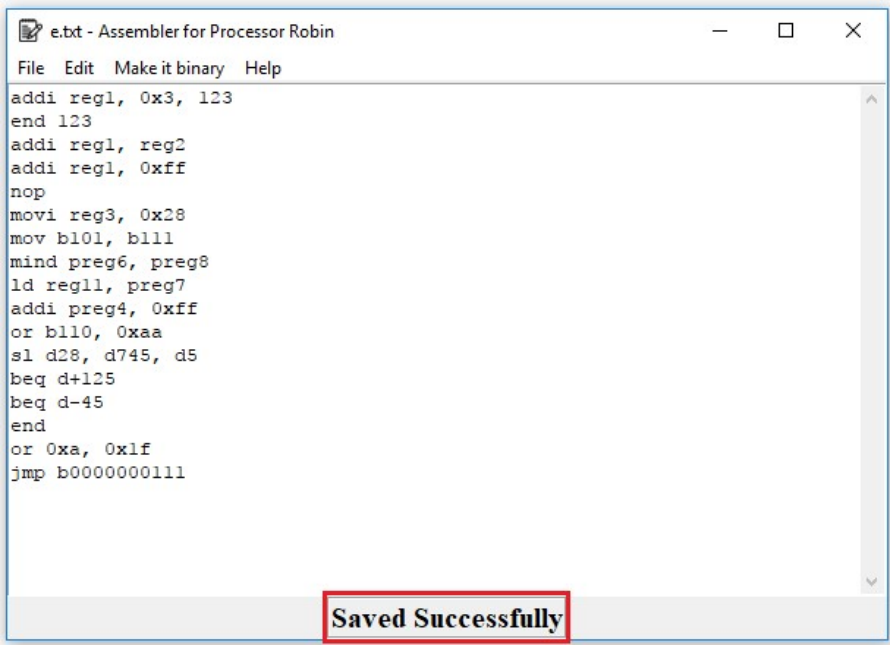
Εικόνα 4.4 - Εμφάνιση περιεχομένου στον χώρο κείμενου

Για την αποθήκευση του αρχείου χρησιμοποιείται το κουμπί «Save». Υπάρχουν δύο περιπτώσεις για την αποθήκευση ενός αρχείου, με κριτήριο εάν το αρχείο υπάρχει ήδη ή έχει δημιουργηθεί εκ νέου.



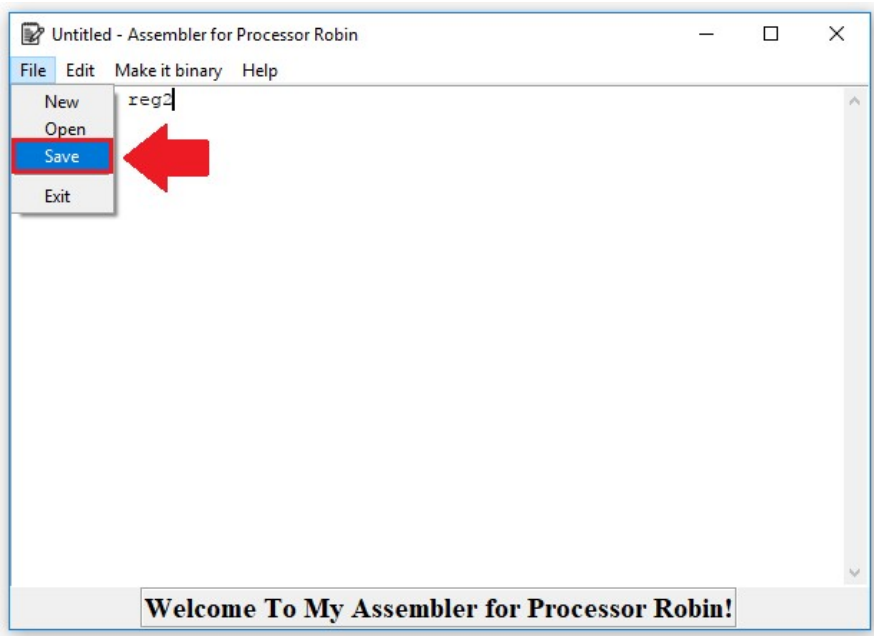
Εικόνα 4.5 - Αποθήκευση του αρχείου

Εάν προϋπάρχει το αρχείο, με το πάτημα του «Save», γίνεται η αποθήκευση του με το ίδιο όνομα στην ίδια τοποθεσία. Ακόμη, εμφανίζεται σχετικό μήνυμα στο κάτω μέρος στο χώρο του status bar.

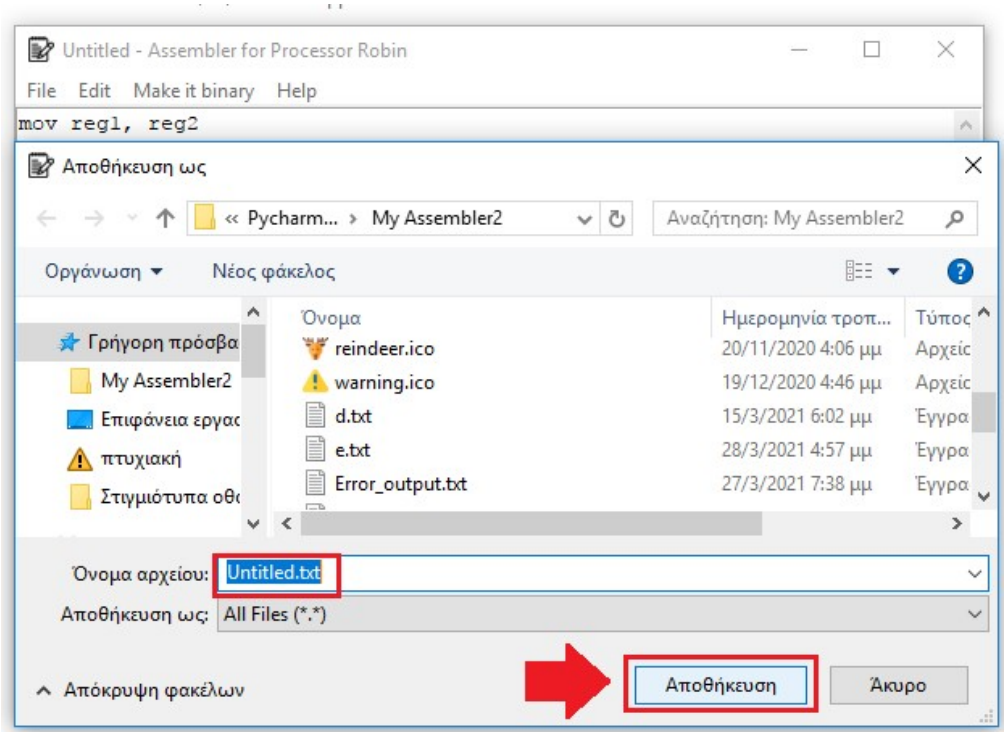


Εικόνα 4.6 - Εμφάνιση μηνύματος επιτυχής αποθήκευσης

Στην περίπτωση που έχει δημιουργηθεί νέο αρχείο όταν χρησιμοποιηθεί το «Save», ανοίγει ένα νέο παράθυρο έτσι ώστε να μπορεί να αλλάξει το όνομα και η τοποθεσία του αρχείου.

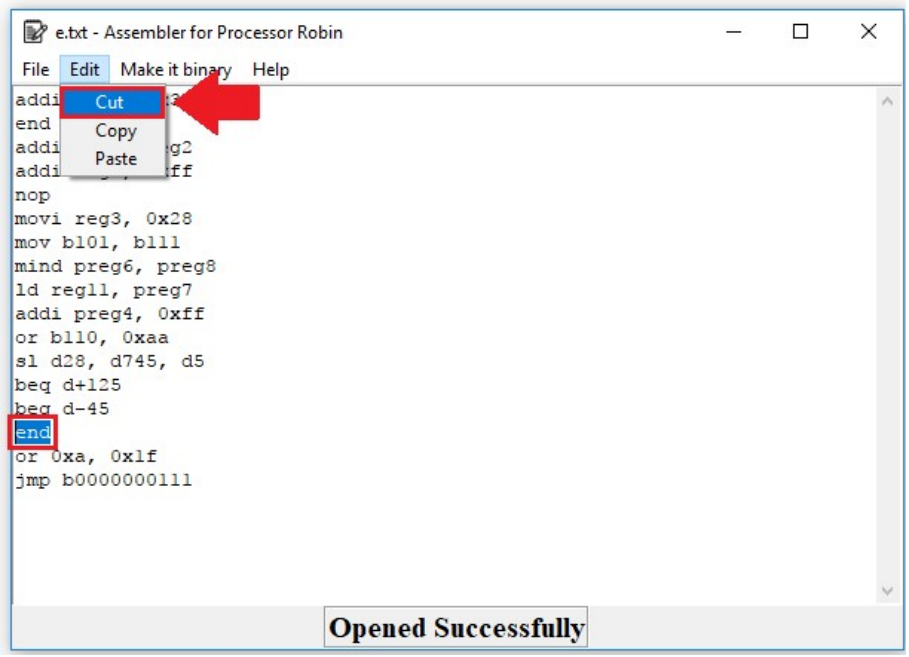


Εικόνα 4.7 - Επιλογή "Save" νέου αρχείου



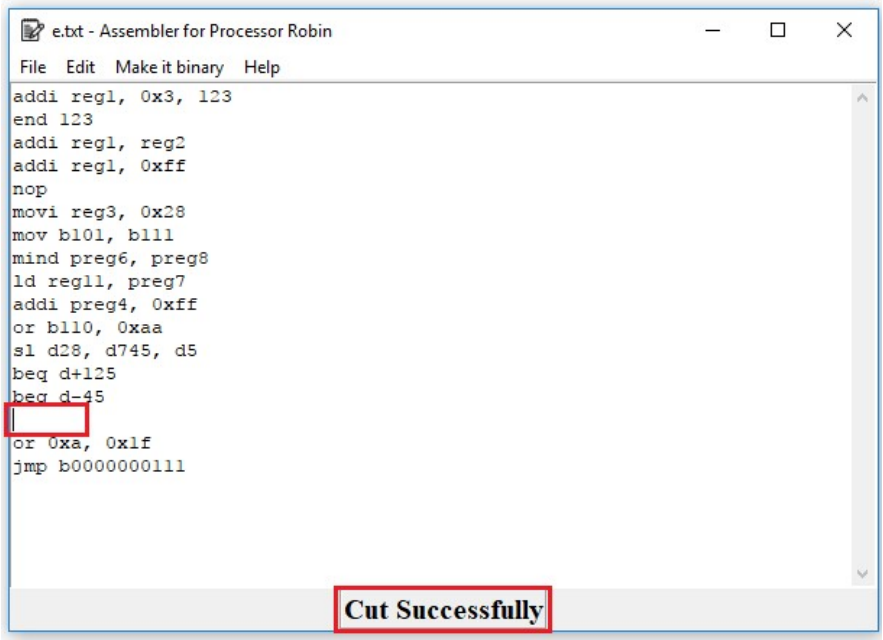
Εικόνα 4.8 - Επιλογή αλλαγής ονόματος του αρχείου και αποθήκευση

Για την επεξεργασία του κειμένου υπεύθυνο είναι το αρχικό μενού «Edit». Με την επιλογή του μενού «Cut», πραγματοποιείται η αποκοπή μιας λέξης από το κείμενο επιλέγοντας τη λέξη με το ποντίκι.



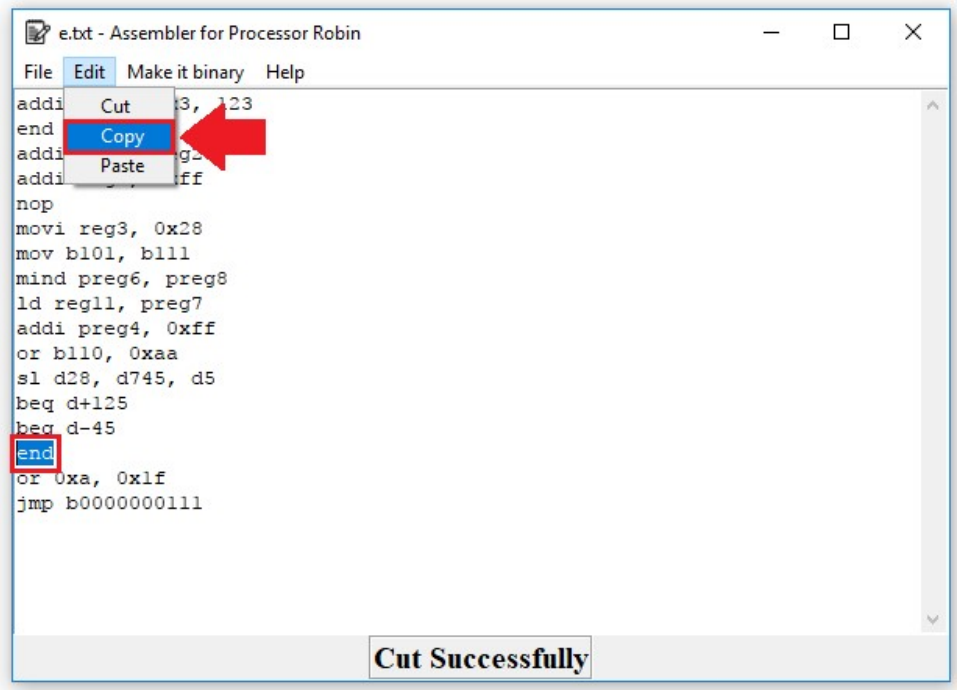
Εικόνα 4.9 – Επιλογή υπομενού "Cut" και επιλογή στοιχείου για αποκοπή

Με την επιτυχή αποκοπή του κειμένου, εμφανίζεται στο status bar σχετικό μήνυμα.



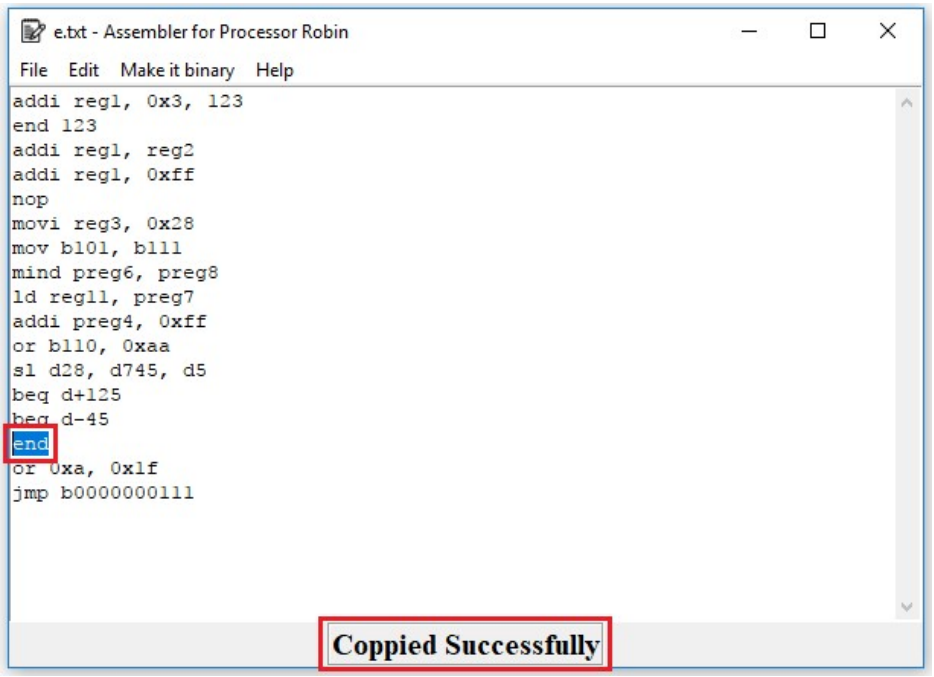
Εικόνα 4.10 - Επιτυχής αποκοπή και εμφάνιση μηνύματος στο status bar

Για την αντιγραφή ενός σημείου του κειμένου, επιλέγεται το κείμενο με το ποντίκι και μέσω του μενού «Copy», ολοκληρώνεται η αντιγραφή.



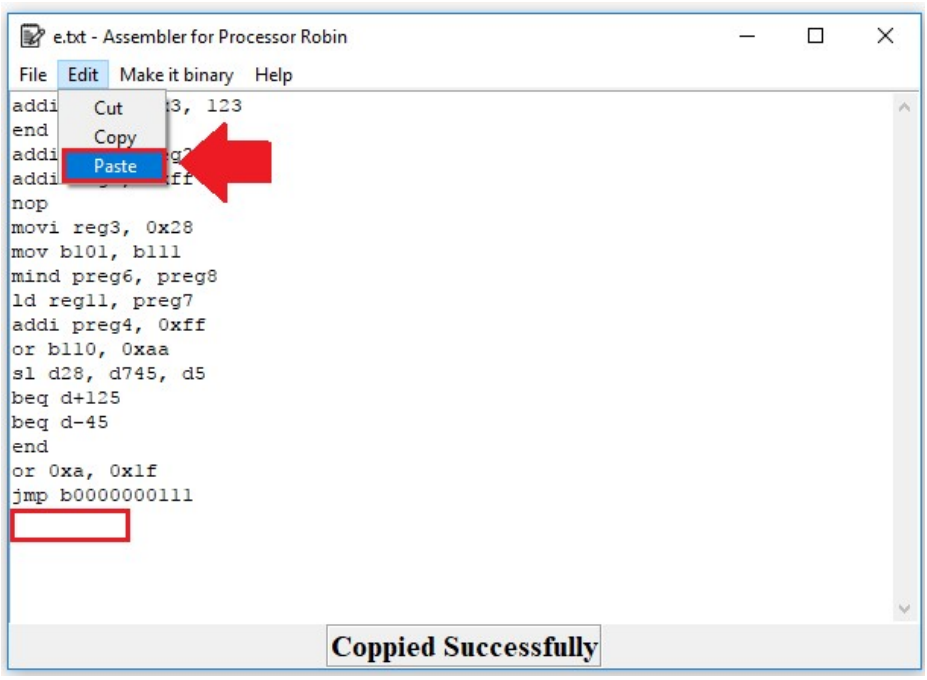
Εικόνα 4.11 – Επιλογή στοιχείου για αντιγραφή

Ενημέρωση του status bar με μήνυμα για την επιτυχία της αντιγραφής.



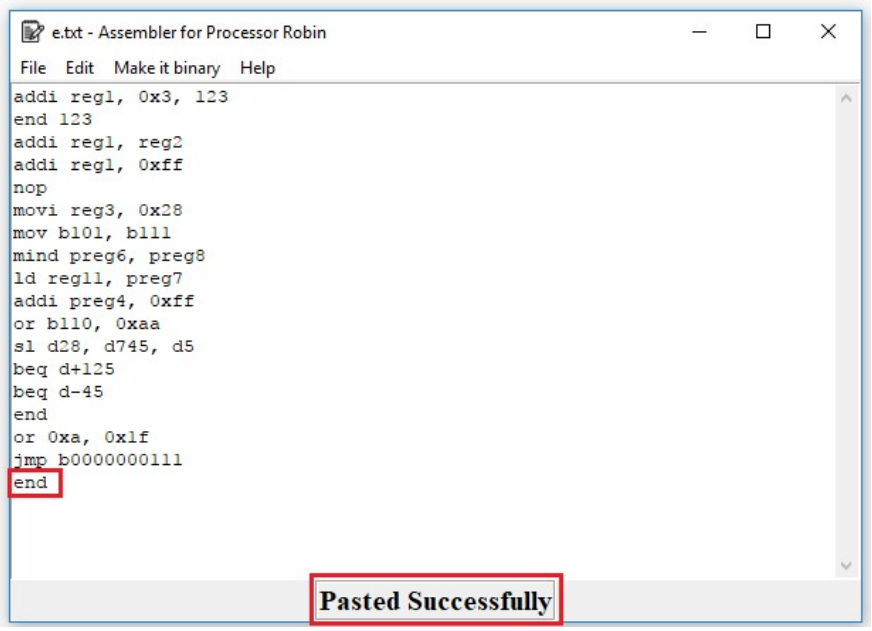
Εικόνα 4.12 - Ένδειξη μέσω του status bar πετυχημένης αντιγραφής

Η επικόλληση του κειμένου πραγματοποιείται βασισμένη στην αντιγραφή που έχει γίνει στο προηγούμενο βήμα. Επιλέγοντας το σημείο επικόλλησης με τον κέρσορα και μέσω του μενού «Paste» εκτελείται η διαδικασία.



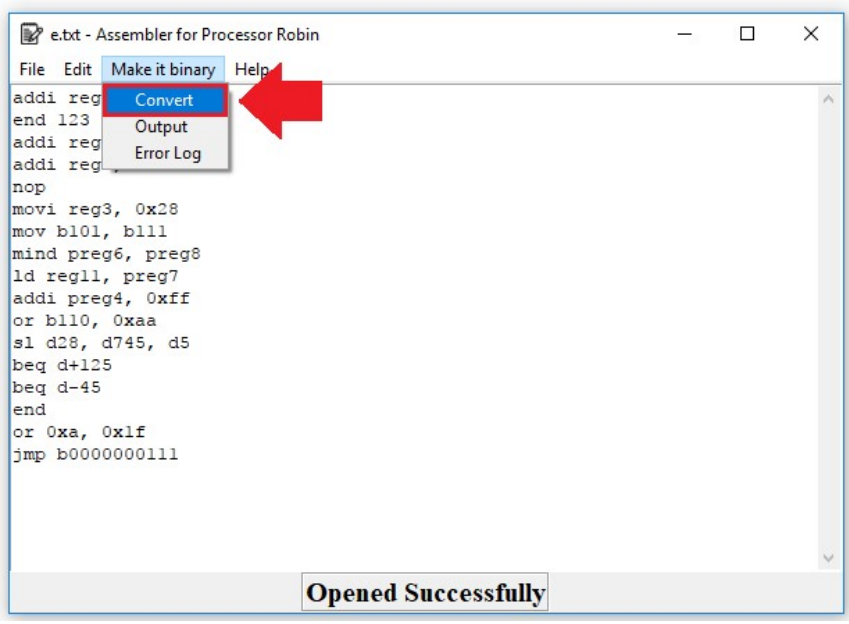
Εικόνα 4.13 - Επιλογή μέσω του κέρσορα το σημείο επιλογής επικόλλησης

Ενημέρωση του status bar με μήνυμα για την επιτυχία της επικόλλησης.



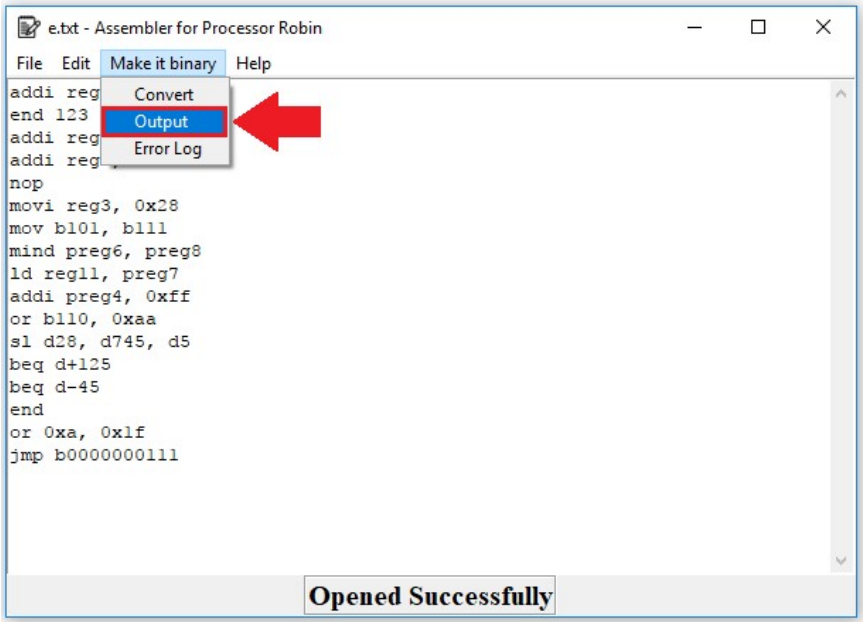
Εικόνα 4.14 - Επιτυχής επικόλληση και εμφάνιση μηνύματος στο status bar

Όταν το αρχείο είναι έτοιμο για μετατροπή σε δυαδική μορφή, γίνεται χρήση του μενού «Convert». Το συγκεκριμένο μενού καλεί και τρέχει το πρόγραμμα του συμβολομεταφραστή. Με την ενέργεια αυτή παράγονται δύο αρχεία εξόδου: το δυαδικό αρχείο εξόδου και αρχείο σφαλμάτων του κώδικα.



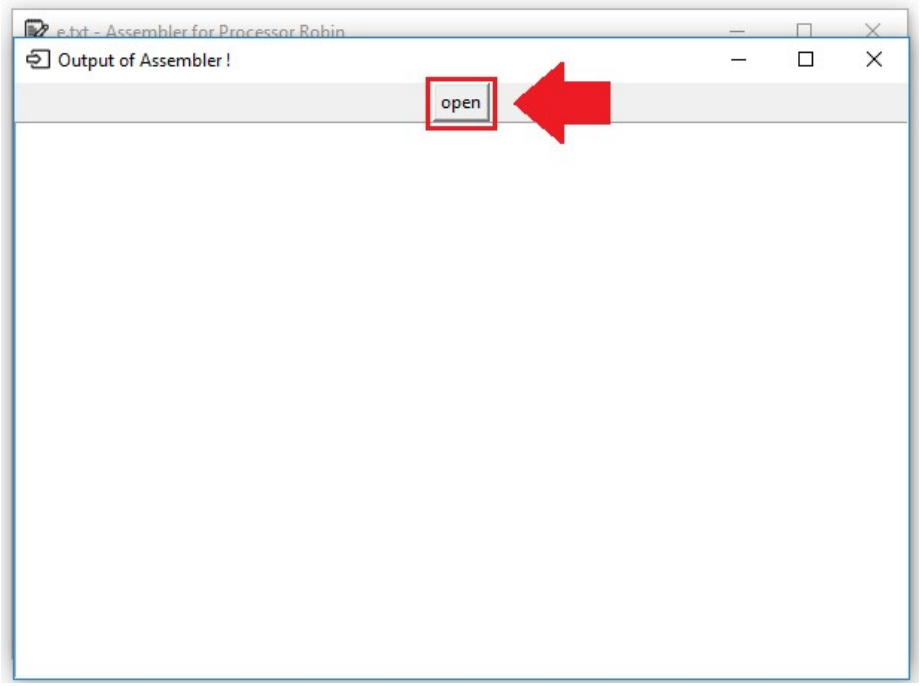
Εικόνα 4.15 - Επιλογή του "Convert" για την μετατροπή του αρχείου

Στη συνέχεια, για να εμφανιστεί το δυαδικό αρχείο γίνεται η επιλογή του μενού «Output», με αυτό τον τρόπο εμφανίζεται ένα νέο παράθυρο.

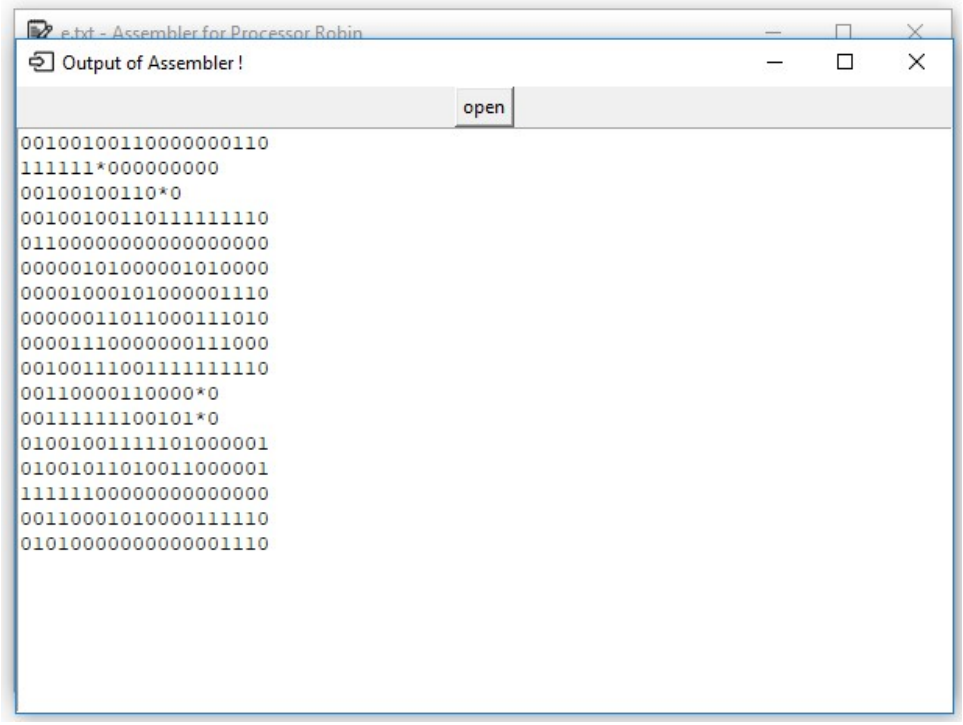


Εικόνα 4.16 - Επιλογή του υπομενού "Output"

Το νέο παράθυρο περιέχει ένα κουμπί που ονομάζεται «open». Η λειτουργία του είναι η εμφάνιση του δυαδικού αρχείου που έχει παραχθεί στον χώρο κειμένου που περιέχει.

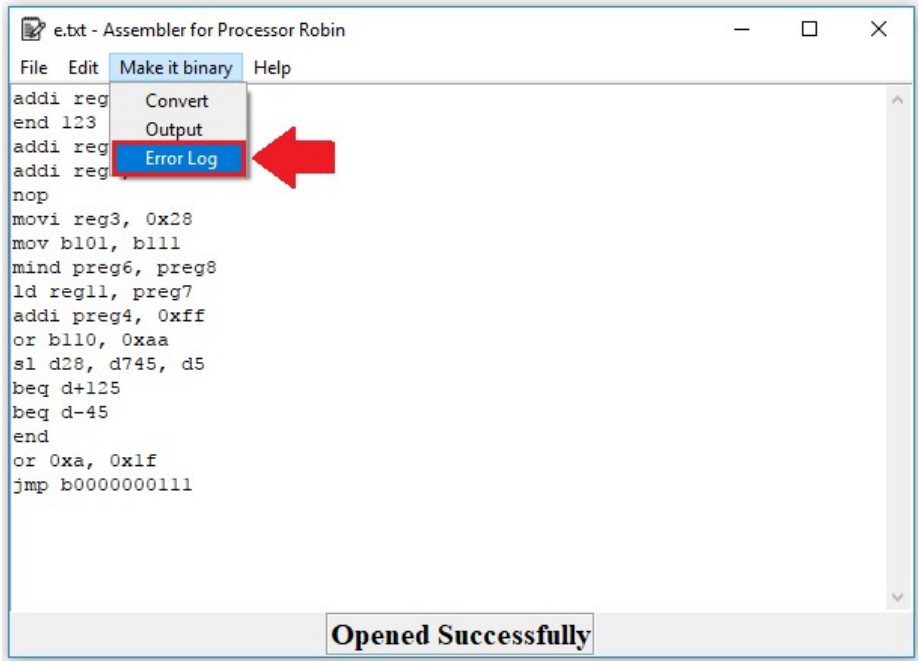


Εικόνα 4.17 - Επιλογή του κουμπιού "open" για το άνοιγμα του δυαδικού αρχείου



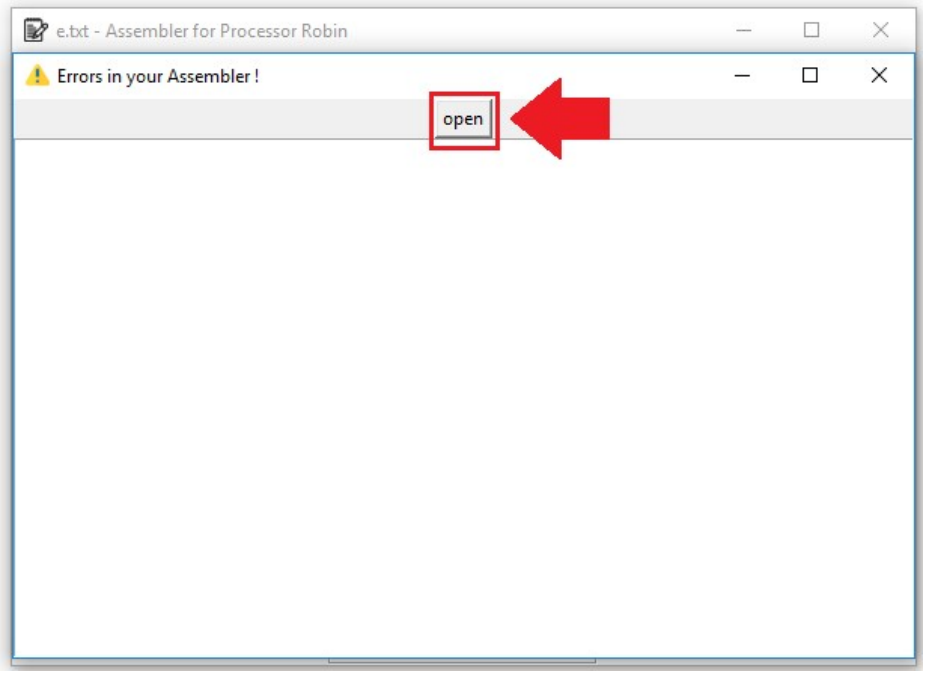
Εικόνα 4.18 – Εμφάνιση του δυαδικού αρχείου

Για τον έλεγχο των σφαλμάτων επιλέγεται το μενού «Error Log».

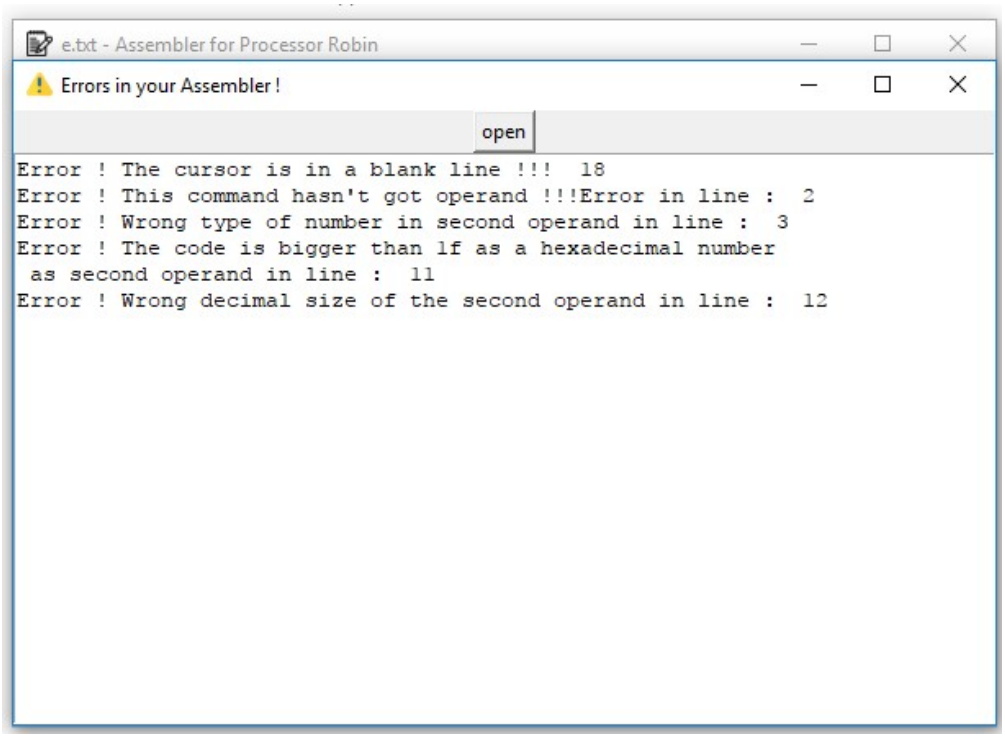


Εικόνα 4.19 - Επιλογή του υπομενού "Error Log"

Με την χρήση του, εμφανίζεται ένα νέο παράθυρο, το οποίο περιέχει ένα κουμπί με το όνομα «open» και μια περιοχή για την εμφάνιση κειμένου. Με το πάτημα του κουμπιού, ανοίγει στην περιοχή κειμένου και εμφανίζει το αρχείο σφαλμάτων.

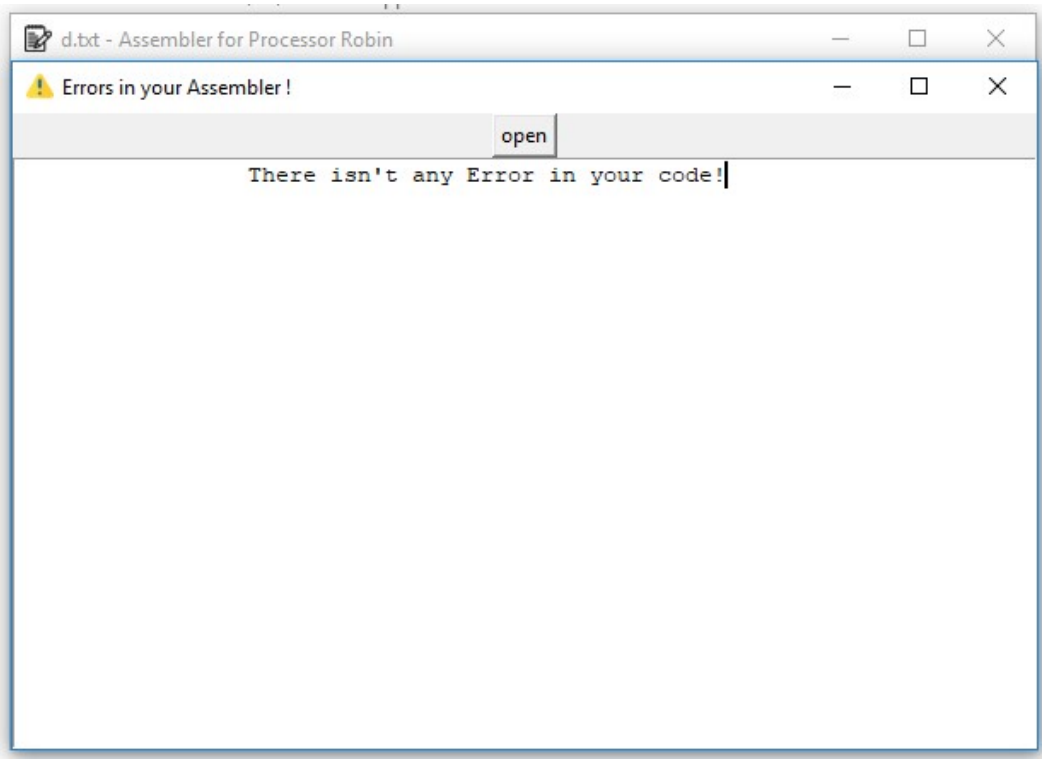


Εικόνα 4.20 - Επιλογή του κουμπιού "open" για άνοιγμα του αρχείου σφαλμάτων



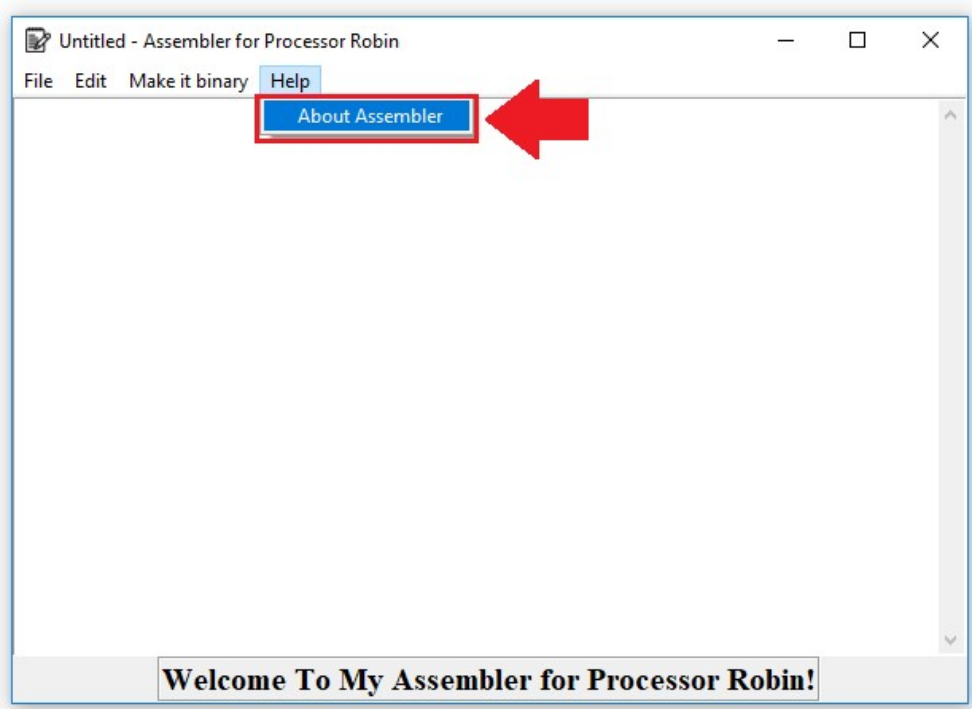
Εικόνα 4.21 - Εμφάνιση σφαλμάτων στον κώδικα του χρήστη

Στην περίπτωση που δεν υπάρχει κάποιο λάθος στον κώδικα εμφανίζεται το σχετικό μήνυμα.



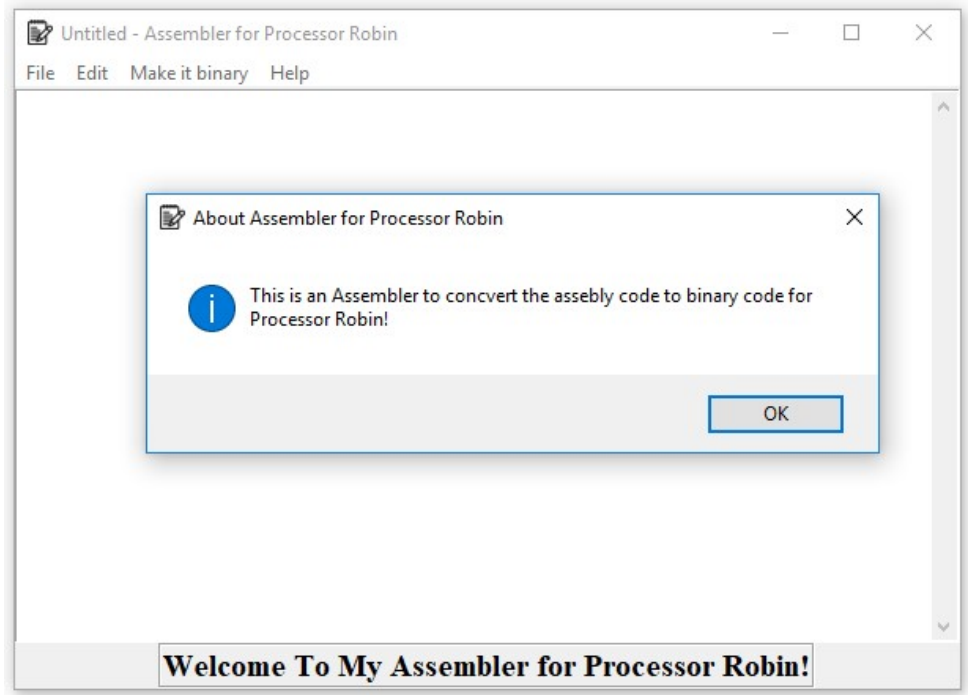
Εικόνα 4.22 - Εμφάνιση μηνύματος στην περίπτωση μηδενικών λαθών

Το τελευταίο μενού ονομάζεται «Help» και περιέχει ένα ακόμα μενού το «About Assembler». Με την χρήση του, εμφανίζεται ένα μήνυμα πληροφόρησης.



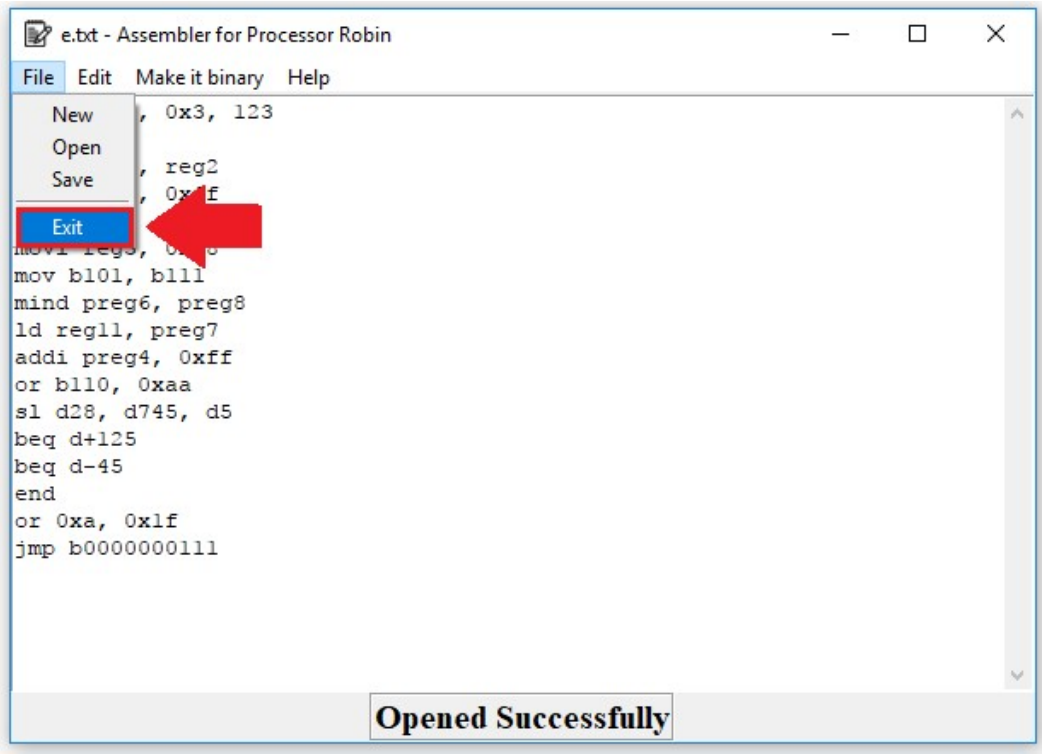
Εικόνα 4.23 - Επιλογή υπομενού "About Assembler"

Με την χρήση του μενού «About Assembler», εμφανίζεται ένα μήνυμα στο χρήστη.



Εικόνα 4.24 - Μήνυμα πληροφόρησης

Τέλος για να κλείσει ο Text Editor, γίνεται επιλογή από το μενού «File» το μενού «Exit» και έτσι κλείνει το παράθυρο.



Εικόνα 4.25 - Κλείσιμο παραθύρου

Κεφάλαιο 5

Συμπεράσματα

Στο τελευταίο κεφάλαιο αναφέρονται τα συμπεράσματα καθώς και κάποιες προτάσεις για βελτίωση της παρούσας πτυχιακής εργασίας.

5.1 Συμπεράσματα και προτάσεις βελτίωσης

Η Python είναι μια γλώσσα προγραμματισμού που εξελίσσεται διαρκώς. Υπάρχουν πολλοί λόγοι, που όλο και περισσότεροι άνθρωποι ασχολούνται με αυτή. Κάποιοι από αυτούς είναι η ραγδαία εξέλιξη της και η ολοένα μεγαλύτερη κοινότητα που αποκτά. Ο κυρίως λόγος της εξέλιξης είναι πως ανανεώνεται και εμπλουτίζεται ο έτοιμος κώδικας που μπορεί να βρει κανείς στο διαδίκτυο.

Βγαίνουν συνεχώς νέες εκδόσεις της Python, στις οποίες προσθέτονται νέα εργαλεία και πιο εμπλουτισμένες βιβλιοθήκες, έτσι ώστε να διευκολύνει τον προγραμματιστή να γράφει όσο το δυνατόν λιγότερο κώδικα με τα ίδια αποτελέσματα και λιγότερα σφάλματα.

5.2 Προτάσεις για μελλοντική έρευνα

Είναι εφικτό να προγραμματίσουμε την παρούσα πτυχιακή εργασία και με άλλες γλώσσες προγραμματισμού όπως η C++ και η JAVA, οι οποίες όμως είναι κάπως πιο πολύπλοκες για τον προγραμματιστή, διότι δεν έχουν τόσες πολλές έτοιμες βιβλιοθήκες.

Οι λειτουργίες του συμβολομεταφραστή μπορούν να γραφούν και με άλλους τρόπους, ή με διαφορετική δομή. Επίσης, όσο αναφορά το γραφικό περιβάλλον υπάρχει η δυνατότητα προγραμματισμού με διαφορετικό framework, εκτός της tkinter. Ακόμα υπάρχει περίπτωση να βρεθούν κι άλλες ανάγκες για προσθήκη στο παράθυρο του Text Editor.

Τέλος, καθώς υπάρχει ραγδαία εξέλιξη της γλώσσας, με την πάροδο του χρόνου χρήσης του προγράμματος ίσως προκύψουν κι άλλες ανάγκες για προσθήκη νέων μενού και διαφοροποίηση κάποιων που ήδη υπάρχουν.

Παράρτημα Α΄

Κώδικας Assembler

Ο παρακάτω κώδικας απαρτίζει τον συμβολομεταφραστή (Assembler). Σε αυτόν ακολουθούν με τη χρήση της δίεσης «#», κάποια σχόλια που βοηθούν στην καλύτερη κατανόηση των τμημάτων του.

```
def assembler():
 import re
 import sys
# Εισαγωγή λεξικού με στοιχεία του από τον Πίνακα 4 και Πίνακα 5
mydict = {"movi": "000001", "mov": "000010", "mind": "000000", "ld": "000011",
"st": "000100", "add": "000101", "sub": "000110", "addc": "000111", "subc":
"001000", "addi": "001001", "subi": "001010", "and": "001011", "or": "001100",
"andi": "001101", "ori": "001110", "sl": "001111", "sr": "010000", "cmp": "010001",
"beq": "010010", "bgt": "010011", "jmp": "010100", "call": "010101", "ret":
"010110", "clr": "010111", "nop": "011000", "end": "111111", "W": "00000","w":
"00000", "status": "00001", "Status": "00001", "IC": "00010", "PCL": "00011",
"PCH": "00100", "Base": "00101", "reg1": "00110", "reg2": "00111", "reg3":
"01000", "reg4": "01001", "reg5": "01010", "reg6": "01011", "reg7": "01100", "reg8":
"01101", "reg9": "01110", "reg10": "01111", "reg11": "10000", "reg12": "10001",
"reg13": "10010", "reg14": "10011", "reg15": "10100", "preg1": "10110", "preg2":
"10111", "preg3": "11000", "preg4": "11001", "preg5": "11010", "preg6": "11011",
"preg7":"11100", "preg8": "11101", "preg9": "11110", "preg10": "11111"}
# Συνάρτηση για τον έλεγχο τιμής εάν είναι 0 ή 1
def isBinary(num: str):
    for i in num:
        if i not in ["0", "1"]:
            return False
    return True
# Έλεγχος του mnemonic εάν ανήκει στην κατηγορία shamt
 def haveshamt(mnem):
    pref = ["sl", "sr"]
    pref = ["movi", "addi", "subi", "addi", "ori", "beq", "bgt" ]
    if mnem.startswith(tuple(pref)):
      return True
```

```
        elif mnem.startswith(tuple(pref)):
            return False

# Έλεγχος του mnemonic εάν ανήκει στην κατηγορία Immediate
    def mnemim(mnemonic):
        pref = ["movi", "addi", "subi", "andi", "ori"]
        if mnemonic.startswith(tuple(pref)):
            return True
        return False
    # Έλεγχος του mnemonic εάν ανήκει στην κατηγορία offset
    def offset(offst):
      pref = ["beq", "bgt"]
      if offst.startswith(tuple(pref)):
        return True
      return False
# Έλεγχος του mnemonic εάν ανήκει στην κατηγορία address
    def address(addrss):
        pref = ["jmp", "call"]
        if addrss.startswith(tuple(pref)):
            return True
        return False
 # Έλεγχος του mnemonic εάν δεν έχει συνέχεια σαν εντολή
    def onlymnemonic(onlymnem):
        pref = ["ret", "nop", "end"]
        if onlymnem.startswith(tuple(pref)):
            return True
        return False
# Άνοιγμα του αρχείου κειμένου "temp.txt" αντιγράφοντας το στο αρχείο "out.txt"
    αντικαθιστώντας το "\n" με κενά πριν από αυτό
    y = open("temp.txt", "rt")
```

```
    x = open("out.txt", "wt")
    for line in y:
      x.write(line.replace('\n', '                    \n'))
    y.close()
    x.close()
    x = open("out.txt", "a")
    x.write("                          \n")
    x.close
#Δημιουργία κενών λιστών για αποθήκευση στοιχείων πριν και μετά την μετατροπή
    list1 = []
    list2 = []
    list3 = []
    list3i = []
    list4 = []
    list5 = []
    fbit = []
    listop1 = []
    listoop1 = []
    listop2 = []
    listoop2 = []
    listop3 = []
#Δημιουργία νέου αρχείου κειμένου στο οποίο θα αποθηκεύονται τα σφάλματα
    Error = open("Error_output.txt", "w")
    sys.stdout = Error
# Άνοιγμα του αρχείου εισόδου , διαχωρισμός της κάθε λέξης σε ξεχωριστή λίστα
    x = open('out.txt')
    for line in x:
      y = re.split(" |,", line)
      list1.append(y[0])
```

```
        list2.append(y[1])
        list3.append(y[3])
        list3i.append(y[5])
# Βρίσκω την αντιστοιχία  mnemonic – opcode  και εμφανίζω σχετικά μηνύματα λάθους σε περίπτωση σφαλμάτων
count = 0
    for z in list1:
      count = count+1
      if z in mydict:
         for k, v in mydict.items():
           if k == z:
              list4.append(v)
              list5.append(k)
      elif z == (""):
         print("Error ! The cursor is in a blank line !!! ", count, file=Error)
      else:
        print("Error ! The command does not exist in the dictionary",
              "!!!ERROR!!! In line : ", count, file=Error)
# Βρίσκω το πρώτο operand
counter = 0
    for zz, mnemonic in zip(list2, list5):
     counter = counter+1
# Έλεγχος εάν δεν έχει operand η εντολή και έχει μόνο mnemonic
if onlymnemonic(mnemonic) == True:
        if not (zz.startswith(" ")):
           if(len(zz) <= 1):
              listop1.append("00000")
           else:
              print("Error ! This command hasn't got operand !!! Error in line : ",
                    counter, file=Error)
              listop1.append("*")
```

```
  # Έλεγχος εάν το operand  είναι της μορφής address .Οι τιμές που μπορεί να πάρει είναι μόνο δυαδικές

elif address(mnemonic) == True:

        if (zz.startswith('b')):

            res1 = (zz[1:])

            if isBinary(res1) == True:

                a = len(res1)

                if (a == 10):

                    s = res1.zfill(13)

                    listop1.append(s)

                else:

                    print("Error ! Wrong size of the first operand in line : ", counter,
                          file=Error)

                    listop1.append("*")

            else:

                print("Error ! Wrong type of number!!! Number of the first operand isn' t
                      binary in line :", counter,file=Error)

                listop1.append("*")

        else:

            print("Error ! The first opcode of  jmp doesn't start with b ", counter ,
                  file=Error)

            listop1.append("*")

  # Έλεγχος εάν το operand  είναι της μορφής offset . Οι τιμές που μπορεί να πάρει είναι δεκαδικές προσημασμένες με όριο τιμών από το -128 έως το +127

elif offset(mnemonic) == True:

            if (zz.startswith('d+')):

                if (len(zz) <= 5):

                    res = (str(zz))

                    res1 = (res[2:])

                    res2 = (int(res1))

                    if res2 <= 127:
```

```
                result = "{0:08b}".format(res2, 10)

                listop1.append(result)

            else:

                print("Error ! The decimal number of the offset is out of
                      boundaries in line : ", counter, file=Error)

                listop1.append("*")

        else:

            print("Error ! Wrong decimal size of the offset in line : ", counter,
                  file=Error)

            listop1.append("*")
```

Εάν είναι αρνητική η τιμή του operand , πριν γίνει η μετατροπή σε δυαδική μορφή μετατρέπεται ως προς συμπλήρωμα ως προς δύο.

```
    elif (zz.startswith('d-')):

        if (len(zz) <= 5):

            res = (str(zz))

            res1 = (res[2:])

            res2 = (int(res1))

            if res2 <= 128:

                m = (256 - res2)

                result = "{0:08b}".format(m, 10)

                listop1.append(result)

            else:

                print("Error ! The decimal number of the offset is out of
                      boundaries in line : ", counter, file=Error)

                listop1.append("*")

        else:

            print("Error ! Wrong decimal size of the offset in line : ", counter,
                  file=Error)

            listop1.append("*")

    else:

        print("Error ! Wrong offset type in line : ", counter, file=Error)
```

```
            listop1.append("*")
# Έλεγχος εάν το mnemonic ανήκει στις εντολές  Immediate  .Οι τιμές που μπορεί να πάρει είναι από το Register File (Πίνακας 1) , δυαδικές ή δεκαεξαδικές.
    elif mnemim(mnemonic) == True:
        if zz in mydict:
            for kk, vv in mydict.items():
                if kk == zz:
                    listoop1.append(kk)
                    listop1.append(vv)
        elif (zz.startswith('b')):
            res = (str(zz))
            res1 = (zz[1:])
            if isBinary(res1) == True:
                a = len(res1)
                if (a <= 5):
                    f = (5 - a)
                    ff = (res1.zfill(f + a))
                    listop1.append(ff)
                else:
                    print("Error ! Wrong size of the first operand in line : ", counter,
                          file=Error)
                    listop1.append("*")
            else:
                print("Error ! Wrong type of number!!! Number of the first operand isn' t
                      binary in line :", counter, file=Error)
                listop1.append("*")
        elif (zz.startswith('0x')):
            if  (len(zz) <= 4):
                res = (str(zz))
                res1 = (res[2:])
                res2 = (str(res1))
```

```
            res3 = int(res2,16)
            if res3 <= 31:
              res4 = "{0:05b}".format(int(res2, 16))
              listop1.append(res4)
            else:
              print("Error ! The code has more than 1f as a hexadecimal number as
                  first operand in line : ", counter, file=Error)
              listop1.append("*")
          else:
            print("Error ! The code has more than 2 digits as a hexadecimal number as
                  first operand in line : ", counter, file=Error)
            listop1.append("*")
        else:
          print("Error ! Wrong number type of the first operand in line : ", counter,
                file=Error)
          listop1.append("*")
# Στην περίπτωση που το mnemonic δεν ανήκει σε κάποια ειδική κατηγορία  μπορεί
το operand να πάρει τιμές από τον Πίνακα 1 , δεκαεξαδικές , δεκαδικές ή δυαδικές.
else:
        if zz in mydict:
          for kk, vv in mydict.items():
            if kk == zz:
              listoop1.append(kk)
              listop1.append(vv)
        elif (zz.startswith('0x')):
          if  (len(zz) <= 4):
            res = (str(zz))
            res1 = (res[2:])
            res2 = (str(res1))
            res3 = int(res2,16)
            if res3 <= 31:
```

```
            res4 = "{0:05b}".format(int(res2, 16))
            listop1.append(res4)
        else:
            print("Error ! The code has more than 1f as a hexadecimal number as
                first operand in line : ", counter, file=Error)
            listop1.append("*")
    else:
        print("Error ! The code has more than 2 digits as a hexadecimal
            number as first operand in line : ", counter, file=Error)
        listop1.append("*")
elif(zz.startswith('d')):
    if (len(zz)<=3):
        res = (str(zz))
        res1 = (res[1:])
        res2 = (int(res1))
        if res2 <= 31:
            result = "{0:05b}".format(int(res1, 10))
            listop1.append(result)
        else:
            print("Error ! The decimal number is out of boundaries in line : ",
                counter, file=Error)
            listop1.append("*")
    else:
        print("Error ! Wrong decimal size of the first operand in line : ",
            counter, file=Error)
        listop1.append("*")
elif(zz.startswith('b')):
    res = (str(zz))
    res1 = (zz[1:])
    if isBinary(res1) == True:
        a = len(res1)
```

```
            if (a <= 5):
              f = (5-a)
              ff = (res1.zfill(f+a))
              listop1.append(ff)
            else:
                print("Error ! Wrong size of the first operand in line : ", counter,
                      file=Error)
                listop1.append("*")
          else:
             print("Error ! Wrong type of number!!! Number of the first operand isn' t
                   binary in line : ", counter, file=Error)
             listop1.append("*")
        elif (zz.startswith("")):
         listop1.append("00000")
# Έλεγχος εάν το  mnemonic του δεύτερου operand ανήκει στην κατηγορία εντολών
Immediate  .Οι τιμές που μπορεί να πάρει είναι δυαδικές  ή δεκαεξαδικές.
    counter = 0
    for zz, mnemonic in zip(list3, list5):
      counter = counter+1
      if mnemim(mnemonic) == True:
         if (zz.startswith('b')):
            res = (str(zz))
            res1 = (zz[1:])
            if isBinary(res1) == True:
               a = len(res1)
               if (a <= 8):
                  f = (8 - a)
                  ff = (res1.zfill(f + a))
                  listop2.append(ff)
               else:
```

```
            print("Error ! Wrong size of the second operand in line : ",
                  counter, file=Error)

            listop2.append("*")

        else:

            print("Error ! Wrong type of number!!! Number of the second operand
                  isn't binary in line :", counter, file=Error)

            listop2.append("*")

    elif (zz.startswith('0x')):

        k = len(zz)

        if k <= 4:

            res = "{0:08b}".format(int(zz, 16))

            listop2.append(res)

        else:

            print("Error ! The code has more than 2 digits as a hexadecimal number
                   as  second operand in line : ", counter, file=Error)

            listop2.append("*")

    elif(zz.startswith('d')):

      if (len(zz)<=4):

        res = (str(zz))

        res1 = (res[1:])

        res2 = (int(res1))

        if res2 <= 255:

            result = "{0:08b}".format(int(res1, 10))

            listop2.append(result)

        else:

            print("Error ! The decimal number is out of boundaries in line : ",
                   counter, file=Error)

            listop2.append("*")
    else:

        print("Error ! Wrong type of number in second operand in line : ",
              counter, file=Error)

        listop2.append("*")
```

```
elif address(mnemonic) == True:
    listop2.append("")
  # Στην περίπτωση που το mnemonic δεν ανήκει σε κάποια ειδική κατηγορία  μπορεί το δεύτερο operand να πάρει τιμές από τον Πίνακα 1 , δεκαεξαδικές , δεκαδικές ή δυαδικές.
else:
      if zz in mydict:
         for kk, vv in mydict.items():
            if kk == zz:
              listoop2.append(kk)
              listop2.append(vv)
      elif (zz.startswith('0x')):
        if  (len(zz) <= 4):
            res = (str(zz))
            res1 = (res[2:])
            res2 = (str(res1))
            res3 = int(res2,16)
            if res3 <= 31:
              res4 = "{0:05b}".format(int(res2, 16))
              listop2.append(res4)
            else:
              print("Error ! The code has more than 1f as a hexadecimal number as
                  second operand in line : ", counter, file=Error)
              listop2.append("*")
        else:
            print("Error ! The code has more than 2 digits as a hexadecimal number as
                    second operand in line : ", counter, file=Error)
            listop2.append("*")
      elif(zz.startswith('d')):
         if (len(zz)<=3):
            res = (str(zz))
```

```
        res1 = (res[1:])
        res2 = (int(res1))
        if res2 <= 31:
            result = "{0:05b}".format(int(res1, 10))
            listop2.append(result)
        else:
            print("Error ! The decimal number is out of boundaries in line : ",
                  counter, file=Error)
            listop2.append("*")
    else:
        print("Error ! Wrong decimal size of the second operand in line : ",
              counter, file=Error)
        listop2.append("*")
elif(zz.startswith('b')):
    res = (str(zz))
    res1 = (zz[1:])
    if isBinary(res1) == True:
        a = len(res1)
        if (a <= 5):
            f = (5-a)
            ff = (res1.zfill(f+a))
            listop2.append(ff)
        else:
            print("Error ! Wrong size of the second operand in line : ", counter,
                  file=Error)
            listop2.append("*")
    else:
        print("Error ! Wrong type of number!!! Number of the second operand
              isn't binary in line : ", counter, file=Error)
        listop2.append("*")
elif (zz.startswith("")):
```

```
            listop2.append("00000")
# Στην περίπτωση που το mnemonic έχει shamt εδώ γίνεται η μετατροπή του . Έχει όριο τιμών από το 0 έως το 7 . Εάν δεν έχει εμφανίζονται τρία μηδενικά στη θέση του
counter = 0
    for zz, mnemonic in zip(list3i, list5):
      counter = counter + 1
      prefixes = ["d0", "d1", "d2", "d3", "d4", "d5", "d6", "d7"]
      if haveshamt(mnemonic) == True:
          if (zz.startswith(tuple(prefixes))):
             if (len(zz) <= 2):
                res = (str(zz))
                res1 = (res[1:])
                result = "{0:03b}".format(int(res1, 10))
                listop3.append(result)
             else:
                print("Error ! Wrong decimal size of the shamt in line : ", counter,
                      file=Error)
                listop3.append("*")
          elif haveshamt(mnemonic) == False:
              listop3.append("")
          elif address(mnemonic) == True:
              listop3.append("")
          else:
            if (len(zz) >= 1):
              print("Error! The specific mnemonic hasn't got shamt in line : ",
                    counter, file=Error)
              listop3.append("*")
            elif (zz.startswith("")):
               listop3.append("000")
# Κλείνει το αρχείο κειμένου με τα σφάλματα , διότι από εδώ και πέρα δεν έχει κάποιο έλεγχο σφαλμάτων
```

```
Error.close()

# Εδώ γίνεται η μετατροπή του Function bit , ανάλογα με το mnemonic θα παίρνει δυαδικές τιμές 0 ή 1.

 for mnem in list5:

     pref1 = ['cmp', 'beq', 'bgt']

     pref0 = ['movi', 'mov', 'mind', 'sl', 'sr', 'call', 'ret', 'clr', 'nop', 'end',  'ld', 'st', 'add',
              'addc', 'subc', 'sub', 'addi', 'subi', 'and', 'or', 'andi', 'ori', 'jmp']
     if mnem in mydict:

         fb1 = mnem.startswith(tuple(pref1))

         fb0 = mnem.startswith(tuple(pref0))

         FB0 = 0

         FB1 = 1

         if fb1 == True:

           fbit.append(FB1)

         elif fb0 == True:

            fbit.append(FB0)

# Δημιουργία αρχείου εξόδου τοποθετώντας σε αυτό με τη σειρά τις εντολές σε δυαδική μορφή. Η σειρά είναι : opcode – operand1 – shamt – operand2 – function bit

    Output = open("output.txt", "w")

    sys.stdout = Output

    for opc, op1, shmt, op2, func_bit in zip(list4, listop1, listop3, listop2, fbit):

       print(opc, op1, shmt, op2, func_bit)

    Output.close()

    yy = open("output.txt", "rt")

    xx = open("output.txt", "wt")

    for line in yy:

       xx.write(line.replace(' ', ''))

    yy.close()

    xx.close()

# Τρέχει η συνάρτηση assembler

if __name__ == '__main__':
```

```
import sys
sys.exit(assembler)
```

Παράρτημα Β΄

Κώδικας για τον Text Editor

```
# Εισαγωγή απαραίτητων βιβλιοθηκών και σύνδεση του Text Editor με άλλα αρχεία
Εισάγοντας συναρτήσεις από αυτά
import os
import tkinter
from tkinter import *
from tkinter.messagebox import *
from tkinter.filedialog import *
from Assembler import assembler
from Errors import errors
from Output import output
# Δημιουργία κλάσης και ορισμός βασικών χαρακτηριστικά του παραθύρου
class Notepad:
    root = Tk()
    thisTextArea = Text(root)
    thisMenuBar = Menu(root)
    thisFileMenu = Menu(thisMenuBar, tearoff=0)
    thisEditMenu = Menu(thisMenuBar, tearoff=0)
    thisCallProg2 = Menu(thisMenuBar, tearoff=0)
    thisHelpMenu = Menu(thisMenuBar, tearoff=0)
    thisNewWindow = Menu(root)
# Εισαγωγή ScrollBar
    thisScrollBar = Scrollbar(thisTextArea)
    file = None
# Χαρακτηριστικά του παραθύρου , Εικόνα , Τίτλος , Μέγεθος , Αυτόματη
Προσαρμογή  Μεγέθους , Δήλωση μεταβλητής Status bar
    def __init__(self, **kwargs):
        try:
```

```
            self.root.wm_iconbitmap(r'notepad.ico')
        except:
            pass
        self.root.title("Untitled - My Assembler for Processor Robin")
        self.root.geometry("600x400")
        self.root.grid_rowconfigure(0, weight=1)
        self.root.grid_columnconfigure(0, weight=1)
        self.thisTextArea.grid(sticky=N + E + S + W)
        self.status = StringVar()
        self.statusbar = Label(self.root, textvariable=self.status,
                               font=("times new roman", 15, "bold"), bd=2, relief=GROOVE)
        self.statusbar.grid(row=2, rowspan=6)
        self.status.set("Welcome To My Assembler for Processor Robin!")
# Ορισμός μενού και υπομενού
# Μενού File
    self.thisFileMenu.add_command(label="New", command=self.newFile)
    self.thisFileMenu.add_command(label="Open", command=self.openFile)
    self.thisFileMenu.add_command(label="Save", command=self.saveFile)
    self.thisFileMenu.add_separator()
    self.thisFileMenu.add_command(label="Exit", command=self.quitApplication)
    self.thisMenuBar.add_cascade(label="File", menu=self.thisFileMenu)
# Μενού Edit
    self.thisEditMenu.add_command(label="Cut", command=self.cut)
    self.thisEditMenu.add_command(label="Copy", command=self.copy)
    self.thisEditMenu.add_command(label="Paste", command=self.paste)
    self.thisMenuBar.add_cascade(label="Edit", menu=self.thisEditMenu)
# Μενού Make it binary
self.thisCallProg2.add_command(label="Convert", command=assembler)
```

```
self.thisCallProg2.add_command(label="Output", command=output)
self.thisCallProg2.add_command(label="Error Log", command=errors)
    self.thisMenuBar.add_cascade(label="Make it binary", menu=self.thisCallProg2)
# Μενού About Assembler
  self.thisHelpMenu.add_command(label="About Assembler",
                                                command=self.showAbout)
    self.thisMenuBar.add_cascade(label="Help", menu=self.thisHelpMenu)
    self.root.config(menu=self.thisMenuBar)
    self.thisScrollBar.pack(side=RIGHT, fill=Y)
# Μενού Αυτόματη προσαρμογή Scrollbar
    self.thisScrollBar.config(command=self.thisTextArea.yview)
    self.thisTextArea.config(yscrollcommand=self.thisScrollBar.set)
# Μενού για δημιουργία νέου αρχείου
 def newFile(self):
        self.root.title("Untitled – Assembler for Processor Robin")
        self.file = None
        self.thisTextArea.delete(1.0, END)
        self.status.set("New File Created")
# Μενού για το άνοιγμα αρχείου
    def openFile(self):
        self.file = askopenfilename(defaultextension=".txt",
                    filetypes=[("All Files", "*.*"), ("Text Documents", "*.txt")])
        if self.file == "":
            self.file = None
        else:
            self.root.title(os.path.basename(self.file) + "-" Assembler for Processor Robin)
            self.thisTextArea.delete(1.0, END)
            file = open(self.file, "rt")
            new_file = open("temp.txt", "wt")
```

```
            with open(self.file, "r") as f:
                new_file.write(f.read())
            new_file.close()
            self.thisTextArea.insert(1.0, file.read())
            self.status.set("Opened Successfully")
            file.close()
    # Μενού για την αποθήκευση του αρχείου
        def saveFile(self):
            if self.file == None:
                self.file = asksaveasfilename(initialfile='Untitled.txt',
                        defaultextension=".txt", filetypes=[("All Files", "*.*"),
                         ("Text Documents", "*.txt")])
                if self.file == "":
                    self.file = None
                else:
                    file = open(self.file, "w")
                    file.write(self.thisTextArea.get(1.0, END))
                    file.close()
                    self.root.title(os.path.basename(self.file) + "-
                                    Assembler for Processor Robin ")
                    self.status.set("Saved Successfully")
            else:
                file = open(self.file, "w")
                file.write(self.thisTextArea.get(1.0, END))
                self.status.set("Saved Successfully")
                file.close()
    # Μενού για το κλείσιμο της εφαρμογής
        def quitApplication(self):
            self.root.destroy()
    # Μενού για αποκοπή
```

```
def cut(self):
        self.thisTextArea.event_generate("<<Cut>>")
        self.status.set("Cut Successfully")

  # Μενού για αντιγραφή
  def copy(self):
        self.thisTextArea.event_generate("<<Copy>>")
        self.status.set("Coppied Successfully")
# Μενού για επικόλληση
    def paste(self):
        self.thisTextArea.event_generate("<<Paste>>")
        self.status.set("Pasted Successfully")
# Μενού για πληροφορίες
    def showAbout(self):
        showinfo("About Assembler ", "This is an Assembler for Processor Robin ! ")
        self.status.set("Opened About")

# Τρέχω την κυρίως εφαρμογή
    def run(self):
        self.root.mainloop()

# Τρέχω το κυρίως πρόγραμμα
notepad = Notepad()
notepad.run()
```

Πρόγραμμα Εμφάνισης Αρχείου Εξόδου

```
def output():
  import tkinter
 # Δημιουργία παραθύρου , Ορισμός μεγέθους , Τίτλου και Εικόνας
    root = tkinter.Tk()
    root.geometry("600x400")
    root.title('Output of Assembler !')
    root.iconbitmap(r'export.ico')

# Συνάρτηση για το άνοιγμα του αρχείου εξόδου από τον Assembler
    def do_open():
        with open("output.txt") as fr:
          content = fr.read()
          text.delete(0.0, tkinter.END)
          text.insert(tkinter.END, content)

# Δημιουργία του κουμπιού «open» που ανοίγει το αρχείο κειμένου
   btn_open = tkinter.Button(root, text='open', command=do_open)
    btn_open.pack()
# Δημιουργία   Text Box
   text = tkinter.Text(root)
    text.pack()
    root.mainloop()
if __name__ == '__main__':
     output()
```

Πρόγραμμα Εμφάνισης Σφαλμάτων

```
def errors():
    import tkinter
    import os

# Δημιουργία παραθύρου , Ορισμός μεγέθους , Τίτλου και Εικόνας
    root = tkinter.Tk()
    root.geometry("600x400")
    root.title('Errors in your Assembler !')
    root.iconbitmap(r'warning.ico')

# Συνάρτηση για το άνοιγμα του αρχείου Σφαλμάτων από τον Assembler
def do_open():
        x = "Error_output.txt"
        with open(x) as fr:
                content = fr.read()
                if os.stat(x).st_size == 0:
            text.insert(tkinter.END, "              There isn't any Error in your code!")
        else:
            text.delete(0.0, tkinter.END)
            text.insert(tkinter.END, content)
  # Δημιουργία του κουμπιού «open» που ανοίγει το αρχείο σφαλμάτων
    btn_open = tkinter.Button(root, text='open', command=do_open)
    btn_open.pack()
# Δημιουργία   Text Box
    text = tkinter.Text(root)
    text.pack()
    root.mainloop()
if __name__ == '__main__':
    errors()
```

Βιβλιογραφία

[1] Ioannis Kalomiros, The Robin SoC processor, MCU Edition: basic specifications

[2] Αριστείδης Σ. Μπούρας , Γιάννης Θ. Κάππος . Python3 Αλγοριθμική και Προγραμματισμός. Εκδόσεις Κλειδάριθμος

[3] Κωνσταντίνος Παπαστεργίου . Μαθαίνω Python & Tkinter. Διαθέσιμο στη διεύθυνση: https://www.ebooks4greeks.gr/mathainw-python-tkinter [τελευταία πρόσβαση 8-12-2021]

[4] Python String startswith Method,
Διαθέσιμο στη διεύθυνση:
https://www.w3schools.com/python/ref_string_startswith.asp
[τελευταία πρόσβαση 16-1-2021]

[5] Python String replace Method,
Διαθέσιμο στη διεύθυνση: https://www.w3schools.com/python/ref_string_replace.asp
[τελευταία πρόσβαση 17-1-2021]

[6] Python String split Method,
Διαθέσιμο στη διεύθυνση: https://www.w3schools.com/python/ref_string_split.asp
[τελευταία πρόσβαση 17-1-2021]

[7] Python String zfill Method ,
Διαθέσιμο στη διεύθυνση: https://www.w3schools.com/python/ref_string_zfill.asp
[τελευταία πρόσβαση 16-1-2021]

[8] Python List append Method,
Διαθέσιμο στη διεύθυνση: https://www.w3schools.com/python/ref_list_append.asp
[τελευταία πρόσβαση 16-1-2021]

[9] Python zip Function,
Διαθέσιμο στη διεύθυνση: https://www.w3schools.com/python/ref_func_zip.asp
[τελευταία πρόσβαση 17-1-2021]

[10] Python len Function,
Διαθέσιμο στη διεύθυνση: https://www.w3schools.com/python/ref_func_len.asp
[τελευταία πρόσβαση 17-1-2021]

[11] Python Dictionaries,
Διαθέσιμο στη διεύθυνση: Python Dictionaries (w3schools.com)
[τελευταία πρόσβαση 11-12-2021]

[12] Check if two lists have any element in common,
Διαθέσιμο στη διεύθυνση: Python - Check if two lists have any element in common (tutorialspoint.com)

[13] Goran Aviani, Python if__name__== __main Explain with Code examples,
Διαθέσιμο στη διεύθυνση: Python if __name__ == __main__ Explained with Code Examples (freecodecamp.org) . [Τελευταία πρόσβαση 11-12-20]

[14] Wikipedia - Harvard architecture,
Διαθέσιμο στη διεύθυνση: https://en.wikipedia.org/wiki/Harvard_architecture
[Τελευταία πρόσβαση 26-12-20]

[15] Wikipedia - Pycharm,
Διαθέσιμο στη διεύθυνση: https://en.wikipedia.org/wiki/PyCharm
[Τελευταία πρόσβαση 27-12-20]

[16] Python Examples - How to Call Function on Tkinter Button Click?,
Διαθέσιμο στη διεύθυνση: https://pythonexamples.org/python-tkinter-button-click-call-function-2/ [Τελευταία πρόσβαση 6-12-20]

[17] Wikipedia - Tkinter,
Διαθέσιμο στη διεύθυνση: https://en.wikipedia.org/wiki/Tkinter
[Τελευταία πρόσβαση 29-12-20]

[18] Python.org – Format String Syntax,
Διαθέσιμο στη διεύθυνση: https://docs.python.org/3/library/string.html#formatspec
[Τελευταία πρόσβαση 10-2-21]

Εικόνες :

Οι εικόνες του πρώτου κεφαλαίου : Εικόνα 1 έως Εικόνα 5 :

πηγή : Ioannis Kalomiros , The Robin SoC processor, MCU Edition: basic specifications

Σχεδιασμός εκ νέου με το εργαλείο : «Google σχέδια» μέσω του «Google drive»

Οι εικόνες - εικονίδια στον Text Editor :

Διαθέσιμο στη διεύθυνση: https://www.flaticon.com/ ,

[Τελευταία πρόσβαση 19-12-20]

Παράθυρο Notepad : Search results for Notepad - Flaticon ,

[Τελευταία πρόσβαση 19-12-20]

Παράθυρο Output : Search results for Export - Flaticon,

[Τελευταία πρόσβαση 19-12-20]

Παράθυρο Errors : https://www.flaticon.com/free-icon/warning_561270?term=warning&page=2&position=10&page=2&position=10&related_id=561270&origin=search ,

[Τελευταία πρόσβαση 19-12-20]