ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

Δημιουργία cycle accurate architectural simulator με χρήση της γλώσσας προγραμματισμού C.

Τριανταφύλλου Ελένη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Σταμούλης Γεώργιος

Λαμία, 2022

UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL BIOMEDICINE

Creating a cycle accurate architectural simulator using the programming language C.

Triantafyllou Eleni

Master thesis

Supervisor
Stamoylis Georgios

Lamia, 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ:

«Πληροφορική με εφαρμογές στην Ασφάλεια, Διαχείριση Μεγάλου Όγκου Δεδομένων και Προσομοίωση»

Δημιουργία cycle accurate architectural simulator με χρήση της γλώσσας προγραμματισμού C.

Τριανταφύλλου Ελένη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Σταμούλης Γεώργιος

Λαμία, 2022

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο «Δημιουργία cycle accurate architectural simulator με χρήση της γλώσσας προγραμματισμού C» αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Η ΔΗΛΟΥΣΑ

Ημερομηνία

Υπογραφή

Δημιουργία cycle accurate architectural simulator με χρήση της γλώσσας προγραμματισμού C.

Τριανταφύλλου Ελένη

Τριμελής Επιτροπή:

Σταμούλης Γεώργιος (επιβλέπων)

……

……

Επιστημονικός Σύμβουλος:

Σταμούλης Γεώργιος

ΠΕΡΙΕΧΟΜΕΝΑ

Περίληψη

Στην ακόλουθη διπλωματική εργασία αναπτύσσουμε μια εφαρμογή προσομοίωσης λειτουργίας και εκτέλεσης υποθετικών εντολών assembly σε περιβάλλον γλώσσας C. Σκοπός είναι να υποστηρίζει ένα σύνολο προκαθορισμένων εντολών, που θα περιλαμβάνουν αριθμητικές πράξεις (πρόσθεση κυρίως), σύγκριση και άλματα προκειμένου να υλοποιηθούν οι κατάλληλες δομές ελέγχου.

Κατά τον σχεδιασμό της αρχιτεκτονικής του προσομοιωτή που αναπτύσσεται, έχει δοθεί ιδιαίτερη προσοχή στην παρακολούθηση των περιεχομένων των καταχωρητών της μνήμης και των στοιχείων που χρησιμοποιούνται και γενικότερα των στοιχείων που χρησιμοποιούνται.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Assembly, Επεξεργαστής, Αλγόριθμοι ομάδες διεργασιών, Εικονικό, Εργαστηριακό, Περιβάλλον, Βελτιστοποίηση απόδοσης, Άλματα, κώδικας τριών διευθύνσεων.

Abstract

In the following dissertation we develop an application for simulating the operation and execution of hypothetical assembly commands in a C. Language environment.

During the design of the simulator architecture that is being developed, special attention has been paid to monitoring the contents of the memory registers and the components used and the components used in general.

KEYWORDS: Assembly, Processor, Process Algorithms, Virtual, Laboratory, Environment, Performance Optimization, Jumps, Three Address Code.

Πρόλογος

Η παρούσα διπλωματική εργασία με τίτλο: «Δημιουργία cycle accurate architectural simulator με χρήση της γλώσσας προγραμματισμού C.» αναπτύχθηκε στο πλαίσιο του διατμηματικού προγράμματος μεταπτυχιακών σπουδών: «Πληροφορική με εφαρμογές στην Ασφάλεια, Διαχείριση Μεγάλου Όγκου Δεδομένων και Προσομοίωση» του Πανεπιστημίου Θεσσαλίας.

Η δημιουργία της εφαρμογής αυτής θα μας δώσει την ευκαιρία να προγραμματίσουμε από το μηδέν έναν νέο επεξεργαστή (στην πλευρά του λογισμικού), σε μία σύγχρονη γλώσσα προγραμματισμού, όπως η γλώσσα C.

Θα κατανοήσουμε καλύτερα την λειτουργία της κάθε μιας εντολής στο παρασκήνιο και τον πραγματικό ρόλο της σε γλώσσα μηχανής. Επίσης στην φάση του σχεδιασμού θα επιλέξουμε της εντολές που θα είναι ενεργές αλλά και αρκετές, ώστε να δοθεί ένα εύρος επιλογών στον χρήστη που θα κληθεί να αναπτύξει τον «προσωπικό του» κώδικα.

Άλλο ένα κίνητρο είναι, μελλοντικά, η εφαρμογή να επεκταθεί με τέτοιο τρόπο, ώστε να δίνεται η επιλογή στους χρήστες της, να κατασκευάσουν εντολές και να τις αποθηκεύσουν σε βιβλιοθήκες προκειμένου να παραμετροποιήσουν προσωπικά την εφαρμογή που θα παρουσιαστεί.

Επίσης μπορούμε να αναπτύξουμε, σε επόμενη φάση, πρόσθετες επιλογές όπως η φόρτωση ενός αρχείου με εντολές, ενός έτοιμου προγράμματος για εκτέλεση και επιπλέον λειτουργίες.

Κεφάλαιο 1ο

1.1 Επισκόπηση διπλωματικής εργασίας

Στο κεφάλαιο αυτό θα αναφερθούμε στην δομή της πτυχιακής εργασίας, δηλαδή θα γίνει αναφορά στα κεφάλαια που την απαρτίζουν, τα περιεχόμενά τους και επιπρόσθετα θα αναφέρουμε στις τεχνικές σχεδίασης που θα ακολουθήσουμε προκειμένου να φτάσουμε στο τελικό αποτέλεσμα.

1.2 Δομή εργασίας

Σε αυτό το σημείο θα κάνουμε μία αναφορά στο περιεχόμενο των κεφαλαίων που θα ακολουθήσουν.

Κεφάλαιο 1: Αντικείμενο και πλάνο υλοποίησης. Στο πρώτο κεφάλαιο γίνεται αναφορά στα κίνητρα και την μεθοδολογία που θα ακολουθήσουμε για την υλοποίηση αυτής της πτυχιακής εργασίας. Θα γίνει μια πρώτη αναφορά στο αρχιτεκτονικό μοντέλο της εφαρμογής μας, καθώς και μία συνοπτική αναφορά στην σύνδεση των δομικών της στοιχείων με την γλώσσα προγραμματισμού C.

Κεφάλαιο 2: Κύριο μέρος πτυχιακής. Περιγράφονται και καθορίζονται, η αρχιτεκτονική της εφαρμογής μας, οι βασικές εντολές που θα χρησιμοποιηθούν και τα διάφορα βήματα της υλοποίησης τους, σύμφωνα πάντα με τον αρχικό μας σχεδιασμό και την δυνατότητα σύνδεσης τους, με την γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε, επίσης τα βοηθητικά υποπρογράμματα τα οποία θα χρησιμοποιηθούν και η λειτουργία τους.

Κεφάλαιο 3: Προγραμματισμός και σύνδεση με την γλώσσα προγραμματισμού C. Στο τρίτο κεφάλαιο πραγματοποιείται ανάλυση του

προβλήματος που καλούμαστε να αντιμετωπίσουμε, προσδιορισμός των απαιτήσεων της εφαρμογής μας και αναφορά στον σχεδιασμό υλοποίησης. Επίσης παρουσιάζεται η σύνδεση των δομικών στοιχείων της εφαρμογής μας με την γλώσσα προγραμματισμού C, καθώς επίσης επιλύονται τα ζητήματα της υλοποίησης και της σύνδεσης των επιμέρους τμημάτων μεταξύ τους, καθορίζονται οι βασικές ενέργειες της εφαρμογής μας καθώς παράλληλα πραγματοποιούνται τυχόν διορθώσεις στα βασικά στοιχεία του αρχικού μας σχεδιασμού.

Κεφάλαιο 4: Πραγματοποιείται η πρώτη μας προσομοίωση και παρατηρούμε τυχόν σφάλματα και ελλείψεις του προσομοιωτή μας, ώστε να εντοπιστούν τυχόν λογικά λάθη στην σχεδίαση μας.

Κεφάλαιο 5: Παρατίθεται ένας οδηγός εγκατάστασης της γλώσσας προγραμματισμού C, ενώ παράλληλα δίνεται ένα παράδειγμα εκτέλεσης στην συγκεκριμένη γλώσσα προγραμματισμού.

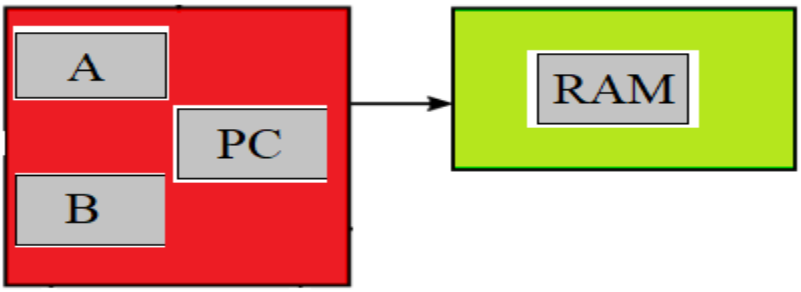
Κεφάλαιο 6: Παρατηρήσεις που έχουν εντοπιστεί, συμπεράσματα από την διαδικασία συγγραφής της πτυχιακής μας και σενάρια μελλοντικών επεκτάσεων.

1.3 Αρχιτεκτονική υποθετικού προσομοιωτή

Στην παρούσα φάση θα πρέπει να οροθετηθούν κάποια όρια στις δυνατότητες της εφαρμογής μας, με σκοπό να γίνει το σύστημα όσον το δυνατόν γίνεται πιο απλό. Το αρχιτεκτονικό μοντέλο του προσομοιωτή μας, έχει οροθετηθεί από τον εισηγητή μας και παρουσιάζεται στο σχήμα 1.1.

Ο υποθετικός προσομοιωτής μας διαθέτει δύο καταχωρητές γενικής χρήσης, τους καταχωρητές A και B και ένα μετρητή προγράμματος (PC) στο τμήμα της εκτέλεσης των εντολών.

Το σύστημα μνήμης θα μπορούσε να είναι ένας πίνακας μονοδιάστατος ο οποίος με την χρήση της γλώσσας C θα μπορούσε να προσομοιωθεί με την δομή δεδομένων του πίνακα. Περισσότερο, αυστηρές προδιαγραφές για το υλικό της εφαρμογής μας θα προσδιοριστεί στο επόμενο κεφάλαιο.



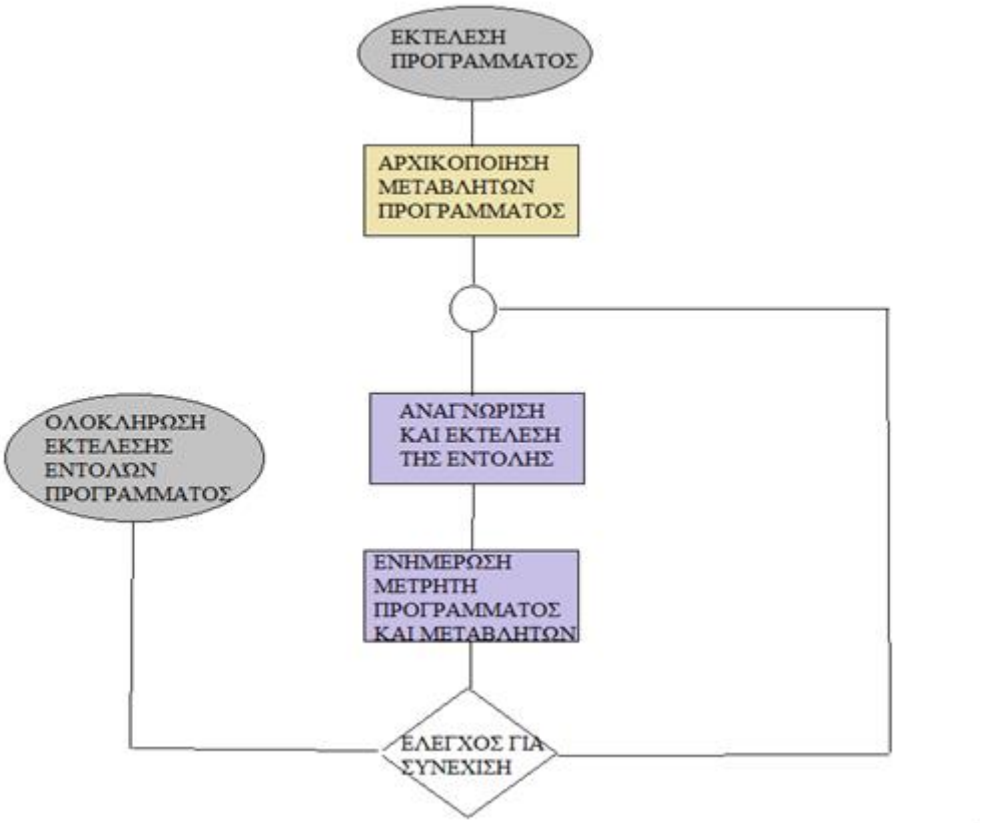
Σχήμα 1.1 Αρχιτεκτονική Υποθετικού Προσομοιωτή

Η βασική ιδέα είναι οι εντολές που δίνει ο χρήστης να αποθηκεύονται σειριακά στην μνήμη και στην συνέχεια, αφού διαβαστούν όλες, να γίνεται η αποκωδικοποίηση τους, να εντοπίζονται τυχόν συντακτικά λάθη, που θα επισημανθούν κατάλληλα και τέλος θα γίνεται η εκτέλεση των εντολών. Στην φάση της μεταγλώττισης και όσον αφορά των χειρισμό των λαθών, θα επιλέγεται η δράση της αντικατάστασης της συγκεκριμένης εντολής, ή της ολικής αρχικοποίησης του υποθετικού προγράμματος. Τελικά οι εντολές θα εκτελεστούν με την κατάλληλη σειρά από την εφαρμογή μας σε γλώσσα C.

Θα δοθεί επίσης έμφαση στην αναγνώριση συντακτικών λαθών που αφορούν δομές ελέγχου (επιλογή και επανάληψη) και στην ανάνηψη τους, αλλά προφανώς τα λογικά λάθη θα μπορούν να υπάρχουν και να αντιμετωπίζονται στην φάση της εκτέλεσης των εντολών του υποθετικού προγράμματος.

1.4 Βασική λειτουργία και εκτέλεση εντολών

Η αποκωδικοποίηση και εκτέλεση των εντολών θα γίνεται βάση της τιμής που είναι αποθηκεμένη στον μετρητή προγράμματος (PC). Αυτός θα δείχνει πάντα την θέση μνήμης της εντολής που πρόκειται να εκτελεστεί και θα είναι αποθηκεμένη στην δομή του πίνακα, και αυτή θα παρομοιάζεται με έναν αριθμό ο οποίος θα αποτελεί τον κώδικα εντολής.



Σχήμα 1.2 Ροή εκτέλεση εντολών Υποθετικού Προσομοιωτή

Ο σχεδιασμός του προσομοιωτή θα επιτρέπει την εκτέλεση της οποιασδήποτε εντολής με κώδικα εντολή διαφορετικό του κώδικα 12, που θα αφορά την εντολή STOP, δηλαδή την εντολή που θα σταματά την εκτέλεση του προγράμματος. Στο σχήμα 1.2 παρουσιάζεται το βασικό διάγραμμα ροής της λειτουργίας του προσομοιωτή μας. Πρόκειται για έναν βρόχο εκτέλεσης εντολών, όπου η ροή εκτέλεσης σταματά μόλις διαβαστεί η εντολή STOP.

Η κάθε υποθετική εντολή αντιστοιχεί και σε μια αντίστοιχη πράξη που εκτελείται, δηλαδή σε μια λειτουργία που έχουμε επιλέξει να γίνεται. Έτσι για παράδειγμα η εντολή INC A θα έχει έναν κώδικα εντολής (πχ 10) όπου και θα αναζητείται στον πίνακα εντολών βάση του κώδικά της, και στην συνέχεια θα πραγματοποιείται αύξηση του περιεχομένου του καταχωρητή (μεταβλητή ακέραια) κατά ένα.

1.5 Η γλώσσα προγραμματισμού που θα χρησιμοποιηθεί

Η γλώσσα προγραμματισμού που θα χρησιμοποιηθεί είναι η γλώσσα C, η οποία μας παρέχει όλα τα απαραίτητα δομικά στοιχεία και εργαλεία για την ανάπτυξη του προσομοιωτή μας. Πρόκειται για μία απλή αλλά παράλληλα δυνατή γλώσσα προγραμματισμού, που θα μας επιτρέψει να αναπτύσσουμε εύκολα και γρήγορα.

Κύριος στόχος της είναι η αυξημένη παραγωγικότητα του προγραμματιστή και η αναγνωσιμότητα του κώδικα. Έχει ενσωματωμένες δομές δεδομένων όπως οι λίστες και οι πίνακες κατακερματισμού και μια μεγάλη βιβλιοθήκη συναρτήσεων.

Η κομψή σύνταξή της και οι δυναμικοί τύποι της, μαζί με την λειτουργία του διερμηνευτή, την καθιστούν ιδανική γλώσσα για δημιουργία σεναρίων εντολών και για ταχεία ανάπτυξη εφαρμογών σε πολλούς τομείς και στις περισσότερες πλατφόρμες.

Εκτεταμένη αναφορά στην γλώσσα προγραμματισμού και στα στοιχεία της που θα χρησιμοποιηθούν, γίνεται στο επόμενο κεφάλαιο.

1.6 Συμπεράσματα

Αναπτύσσοντας τον ίδιο τον προσομοιωτή , θα μπούμε στην διαδικασία σχεδίασης και περιγραφής ενός υποθετικού υπολογιστή. Αυτό θα μας δώσει την δυνατότητα να κατανοήσουμε πλήρως τα δομικά του στοιχεία και χαρακτηριστικά καθώς επίσης και την λειτουργία του, αφού θα μας βάλει στην διαδικασία να σκεφτούμε προδιαγραφές, περιορισμούς που προτάσσει η γλώσσα προγραμματισμού που επιλέγεται, στοιχεία που θα πρέπει να μελετηθούν και να προσπεραστούν, με τις δυνατότητες μιας γλώσσας προγραμματισμού υψηλού επιπέδου.

Κεφάλαιο 2ο

2.1 Εισαγωγή

Η γλώσσα προγραμματισμού C είναι μια γλώσσα γενικού σκοπού, υψηλού επιπέδου που αρχικά αναπτύχθηκε από τον Dennis M. Ritchie για την ανάπτυξη του λειτουργικού συστήματος UNIX. Η γλώσσα εφαρμόστηκε αρχικά στον υπολογιστή DEC PDP-11 το 1972.

Το 1978, ο Μπράιαν Κέρνιγκαν και ο Ντένις Ρίτσι δημιούργησαν το πρώτη δημόσια διαθέσιμη περιγραφή της γλώσσας C, τώρα γνωστό ως πρότυπο K&R.

Το λειτουργικό σύστημα UNIX, ο μεταγλωττιστής C και ουσιαστικά όλα τα προγράμματα εφαρμογών UNIX έχουν γραφτεί σε γλώσσα προγραμματισμού C. Η γλώσσα έχει γίνει πλέον μια ευρέως χρησιμοποιούμενη επαγγελματική γλώσσα για ποικίλους λόγους:

- Εύκολη στην εκμάθηση
- Δομημένη γλώσσα
- Παράγει αποτελεσματικά προγράμματα.
- Μπορεί να χειριστεί δραστηριότητες χαμηλού επιπέδου.
- Μπορεί να καταρτιστεί σε διάφορες πλατφόρμες υπολογιστών.

Σήμερα, η C είναι η πιο διαδεδομένη και δημοφιλής γλώσσα προγραμματισμού συστήματος καθώς τα περισσότερα από τα υπερσύγχρονα λογισμικά έχουν εφαρμοστεί χρησιμοποιώντας C. Παραδείγματα χρήσης του γλώσσας προγραμματισμού C είναι:

- Λειτουργικά συστήματα

- Μεταγλωττιστές γλωσσών
- Επεξεργαστές κειμένου
- Προγράμματα οδήγησης δικτύου
- Σύγχρονα προγράμματα
- Βάσεις δεδομένων
- Διερμηνείς γλωσσών

2.2 Τα χαρακτηριστικά της γλώσσας C

Η γλώσσα προγραμματισμού C είναι μια γλώσσα προγραμματισμού ευρέως χρησιμοποιούμενη που χαρακτηρίζεται ως γλώσσα γενικού σκοπού (general-purpose), για την δημιουργία οποιουδήποτε είδους προγράμματος, ενώ παράλληλα και υψηλού επιπέδου (high-level programming language) καθώς υποστηρίζει πολλά διαφορετικά πρότυπα προγραμματισμού (αντικειμενοστραφής, δυναμική, .ο.κ).

Λόγω του ευσύνοπτου κώδικά της έγινε γρήγορα δημοφιλής και εξελίχθηκε ώστε σήμερα να περιγράφεται γενικότερα με τα παρακάτω χαρακτηριστικά:

Απλή: Η γλώσσα προγραμματισμού C είναι μια απλή και 'μινιμαλιστική' γλώσσα. Το διάβασμα ενός καλού προγράμματος γραμμένο σε C είναι σαν το διάβασμα οδηγιών σε φυσική γλώσσα. Η σύνταξή της μοιάζει αρκετά με την σύνταξη του ψευδοκώδικα, χαρακτηριστικό που αποτελεί ένα από τα πιο ισχυρά σημεία της.

Εύκολη εκμάθηση: Λόγο της απλότητας της, και του χαλαρού τρόπου σύνταξης, επιτρέπει την εύκολη και γρήγορη εκμάθησή της.

Ελεύθερη και Ανοικτού Κώδικα: Η γλώσσα προγραμματισμού C είναι ένα τυπικό παράδειγμα. Με απλά λόγια, μπορείτε να διανείμετε αντίγραφα

αυτού του λογισμικού, να διαβάσετε τον πηγαίο κώδικά του, να κάνετε αλλαγές σ' αυτό και να χρησιμοποιήσετε κομμάτια του σε νέα ελεύθερα προγράμματα.

Γλώσσα υψηλού επιπέδου: Όταν γράφετε προγράμματα στην C, δε χρειάζεται ποτέ να νοιάζεστε για τις χαμηλού επιπέδου λεπτομέρειες όπως η διαχείριση της μνήμης που χρησιμοποιείται από τα προγράμματά σας.

Αν και αυτό πολλοί το θέτουν και ως μειονέκτημα, στην πραγματικότητα έχουν αναπτυχτεί βιβλιοθήκες διασύνδεσης με γλώσσες χαμηλού επιπέδου. Έτσι δίνεται η δυνατότητα στον χρήστη ανάλογα με τις απαιτήσεις να διευθετήσει λεπτομέρειες όπως η διαχείριση της μνήμης, σαν να γράφει με γλώσσα χαμηλού επιπέδου.

Φορητή: Λόγω του ανοικτού της κώδικα, η C έχει υλοποιηθεί (αλλάχθηκε για να λειτουργεί) σε πολλές πλατφόρμες λειτουργικών συστημάτων. Όλα τα προγράμματά που είναι γραμμένα σε C, μπορούν να δουλέψουν σε οποιαδήποτε από αυτές τις πλατφόρμες λειτουργικών συστημάτων, χωρίς να χρειάζονται καθόλου αλλαγές αν είστε αρκετά προσεκτικοί ώστε να αποφύγετε να χρησιμοποιήσετε χαρακτηριστικά που εξαιρούνται κάθ' έκαστο λειτουργικό σύστημα.

Μπορείτε να χρησιμοποιήσετε την γλώσσα προγραμματισμού C στο Linux, στα Windows, στο FreeBSD, σε Macintosh, στο Solaris, στο OS/2, στην Amiga, στο AROS, στο AS/400, στο BeOS, στο OS/390, στο z/OS, στο Palm OS, στο QNX, στο VMS, στο Psion, στο Acorn RISC OS, στο VxWorks, σε PlayStation, στο SharpZaurus, στα Windows CE.

Διερμηνευόμενη: Ένα πρόγραμμα που γράφεται σε μια μεταγλωττιζόμενη γλώσσα όπως η C ή η C++ μετατρέπεται σε πηγαία γλώσσα μηχανής, (δυαδικός κώδικας δηλαδή 0 και 1), χρησιμοποιώντας ένα μεταγλωττιστή με διάφορες σημαίες και επιλογές. Όταν τρέχετε το πρόγραμμα, ο συνδετής αντιγράφει το πρόγραμμα στη μνήμη και αρχίζει να το τρέχει.

Επεκτάσιμη: Αν χρειάζεστε ένα κρίσιμο κομμάτι κώδικα να τρέχει πολύ γρήγορα ή αν πρέπει να έχετε ένα κομμάτι ενός αλγόριθμου που να μην είναι ανοικτό, τότε μπορείτε να προγραμματίσετε εκείνο το κομμάτι σε C \ C++ ή java ή ακόμη και Cuda και μετά να το χρησιμοποιείτε από το πρόγραμμά σας διατηρώντας τα περισσότερα από τα πλεονεκτήματα των άλλων γλωσσών.

2.3 Δομικά στοιχεία της γλώσσας C

Στην συγκεκριμένη ενότητα θα αναφερθούμε στις μεταβλητές που μπορούμε να χρησιμοποιούμε στον προγραμματισμό με την γλώσσα C και τον τρόπο με τον οποίο αυτές δηλώνονται. Επίσης θα γίνει αναφορά και στους τελεστές που μπορούμε να χρησιμοποιήσουμε και χρησιμοποιούμε στην ανάπτυξη της εφαρμογής μας.

2.3.1 Δομή προγράμματος σε γλώσσα C

Ας δούμε το παράδειγμα Hello World χρησιμοποιώντας τη γλώσσα προγραμματισμού C. Πριν μελετήσουμε βασικά δομικά στοιχεία της γλώσσας προγραμματισμού C, ας δούμε την ελάχιστη δομή προγράμματος C, ώστε να μπορούμε να το λάβουμε ως αναφορά στο επόμενο κεφάλαιο.

Ένα πρόγραμμα C αποτελείται βασικά από τα ακόλουθα μέρη:

1. Εντολές Προεπεξεργαστή
2. Λειτουργίες
3. Μεταβλητές
4. Δηλώσεις & Εκφράσεις
5. Σχόλια

Ας δούμε έναν απλό κώδικα που θα εκτυπώνει τις λέξεις "Hello World":

```c
#include <stdio.h>

int main()
{
   /* my first program in C */
   printf("Hello, World! \n");

   return 0;
}
```

Ας δούμε διάφορα μέρη του παραπάνω προγράμματος:

1. Η πρώτη γραμμή του προγράμματος #include <stdio.h> είναι μια εντολή προεπεξεργαστή, η οποία λέει ένας μεταγλωττιστής C για να περιλαμβάνει αρχείο stdio.h πριν μεταβείτε στην πραγματική μεταγλώττιση.
2. Η επόμενη γραμμή int main () είναι η κύρια συνάρτηση όπου ξεκινά η εκτέλεση του προγράμματος.
3. Η επόμενη γραμμή /*...*/ θα αγνοηθεί από τον μεταγλωττιστή και έχει τεθεί για προσθήκη επιπλέον σχόλια στο πρόγραμμα. Έτσι, τέτοιες γραμμές ονομάζονται σχόλια στο πρόγραμμα.
4. Η επόμενη γραμμή printf (...) είναι μια άλλη λειτουργία διαθέσιμη στο C που προκαλεί το μήνυμα "Γειά σου Κόσμε!" να εμφανιστεί στην οθόνη.
5. Η επόμενη γραμμή επιστρέφει 0. τερματίζει τη συνάρτηση main () και επιστρέφει την τιμή 0.

2.3.2 Μεταγλώττιση & Εκτέλεση προγράμματος C

Ας δούμε πώς να αποθηκεύσετε τον πηγαίο κώδικα σε ένα αρχείο και πώς να το μεταγλωττίσετε και να το εκτελέσετε σε απλά βήματα:

1. Ανοίξτε έναν επεξεργαστή κειμένου και προσθέστε τον παραπάνω κωδικό.
2. Αποθηκεύστε το αρχείο ως hello.c
3. Ανοίξτε μια γραμμή εντολών και μεταβείτε στον κατάλογο στον οποίο αποθηκεύσατε το αρχείο.

4. Πληκτρολογήστε gcc hello.c και πατήστε enter για να μεταγλωττίσετε τον κωδικό σας.
5. Εάν δεν υπάρχουν σφάλματα στον κωδικό σας, η γραμμή εντολών θα σας μεταφέρει στην επόμενη γραμμή και θα δημιουργούσε ένα εκτελέσιμο αρχείο a.out.
6. Τώρα, πληκτρολογήστε a.out για να εκτελέσετε το πρόγραμμά σας.
7. Θα μπορείτε να δείτε το "Hello World" τυπωμένο στην οθόνη

```
$ gcc hello.c
$ ./a.out
Hello, World!
```

Βεβαιωθείτε ότι ο μεταγλωττιστής gcc βρίσκεται στη διαδρομή σας και ότι τον εκτελείτε στον κατάλογο που περιέχει το αρχείο προέλευσης hello.c

2.4 Μεταβλητές της γλώσσας C

Μεταβλητές είναι ο χώρος στην μνήμη που αποθηκεύονται προσωρινά δεδομένα. Στην γλώσσα C χωρίζουμε τα είδη των μεταβλητών ανάλογα με το εμβέλεια δράσης τους σε τοπικές και Γενικές (Global) μεταβλητές.

2.4.1 Τοπικές μεταβλητές

Όταν μία μεταβλητή δηλώνεται μέσα σε μία ορισμένη συνάρτηση, αυτή δεν έχει καμία σχέση με οποιαδήποτε άλλη μεταβλητή εκτός της συνάρτησης ακόμη κι αν έχει την ίδια ονομασία και χρησιμοποιείται έξω από αυτή τη συνάρτηση, δηλαδή η χρησιμότητα των μεταβλητών περιορίζεται τοπικά εντός της συνάρτησης.

2.4.2 Γενικές (Global) μεταβλητές

Είναι κάθε εντολή εκτός συνάρτησης. Εκτός αν μια μεταβλητή ορισθεί μέσα σε κάποια συνάρτηση αλλά οριστεί ως γενική, αυτό γίνεται με τη χρήση της εντολής global. Χρησιμοποιώντας την εντολή global γίνεται ξεκάθαρο ότι η μεταβλητή μπορεί να χρησιμοποιηθεί από κάθε εντολή βρίσκεται εντός ή εκτός της προκείμενης συνάρτησης.

2.4.3 Τύποι μεταβλητών της γλώσσας και οι τελεστές τους

Γίνεται αναφορά μόνο στους τύπους μεταβλητών που έχουμε χρησιμοποιήσει για την υλοποίηση της εφαρμογής.

2.4.3.1 Τύπος Integer - ακέραιος

Στην γλώσσα C οι ακέραιοι είναι θεωρητικά άπειρης ακρίβειας. Η γλώσσα C δεσμεύει δυναμικά τον απαραίτητο χώρο μνήμης για την αναπαράσταση οσοδήποτε μεγάλων ακεραίων. Προσοχή όμως στο χρόνο που χρειάζεται για υπολογισμούς με τέτοιους μεγάλους αριθμούς, πχ. ο 2 στην 1000000 δύναμη έχει 301030 ψηφία.

Οι τελεστές που μπορούν να εφαρμοστούν σε αριθμητικά δεδομένα είναι οι:

- Πρόσθεση +
- Αφαίρεση -
- Πολ/σμός *
- Διαίρεση /
- Υπόλοιπο Διαίρεσης ακεραίων %

2.4.3.2 Τύπος χαρακτήρα – Char

Ο τύπος char, έχει μέγεθος 8 bits: αναπαριστά 2^8 διαφορετικές τιμές, συνήθως ASCII χαρακτήρες. Οι τιμές αλφαριθμητικών μπορούν να περικλείονται σε μονά εισαγωγικά πχ. ‘Spam’ είναι αποδεκτά.

Οι συμβολοσειρές (ή ‘αλφαριθμητικά’ εφόσον περιλαμβάνουν γράμματα και αριθμητικά ψηφία) χρησιμοποιούνται για την αποθήκευση κειμένου (πχ. ονόματα, κλπ). Οι τιμές αλφαριθμητικών μπορούν να περικλείονται διπλά εισαγωγικά πχ. “Spam” είναι αποδεκτά.

Τα αλφαριθμητικά είναι ένα χαρακτηριστικό παράδειγμα δομής ακολουθίας (sequence) γλώσσα C, δηλαδή αποτελεί μια δεικτοδοτημένη (indexed) συλλογή αντικειμένων. Μια ακολουθία οργανώνει τα αντικείμενα που την αποτελούν σε σειρά από αριστερά προς τα δεξιά και η θέση τους προσδιορίζεται με έναν δείκτη (index). Ένα αλφαριθμητικό λοιπόν είναι μια ακολουθία χαρακτήρων.

Η υλοποίηση τους στην εφαρμογή μας γίνεται με την βοήθεια του πίνακα που θα αναπτύξουμε σε επόμενη ενότητα μας.

2.4.3.3 Οι υπόλοιποι τελεστές της γλώσσας C

Η παρακάτω συσχέτιση δείχνει όλους τους τελεστές σχέσης που υποστηρίζονται από τη γλώσσα C. Ας υποθέσουμε ότι η μεταβλητή A έχει την τιμή 10 και η μεταβλητή B έχει την τιμή 20. Παράδειγμα περιγραφής τελεστή:

== Ελέγχει εάν οι τιμές δύο τελεστών είναι ίσες ή όχι, αν ναι τότε η συνθήκη γίνεται αληθής. (A == B) δεν ισχύει.

!= Ελέγχει εάν οι τιμές δύο τελεστών είναι ίσες ή όχι, εάν οι τιμές δεν είναι ίσες τότε η συνθήκη γίνεται αληθής. (A! = B) είναι αλήθεια.

> Ελέγχει εάν η τιμή του αριστερού τελεστή είναι μεγαλύτερη από την τιμή του δεξιού τελεστή, εάν ναι τότε η συνθήκη γίνεται αληθής. (A> B) δεν ισχύει.

< Ελέγχει εάν η τιμή του αριστερού τελεστή είναι μικρότερη από την τιμή του δεξιού τελεστή, αν ναι τότε η συνθήκη γίνεται αληθής. (A <B) είναι αλήθεια.

>= Ελέγχει εάν η τιμή του αριστερού τελεστή είναι μεγαλύτερη ή ίση με την τιμή του δεξιού τελεστή, εάν ναι τότε η συνθήκη γίνεται αληθής. (A> = B) δεν ισχύει.

<= Ελέγχει εάν η τιμή του αριστερού τελεστή είναι μικρότερη ή ίση με την τιμή του δεξιού τελεστή, αν ναι τότε η συνθήκη γίνεται αληθής. (A <= B) είναι αλήθεια.

Επίσης χρησιμοποιήθηκαν και οι αριθμητικοί τελεστές:

++ Ο τελεστής αυξήσεων αυξάνει την ακέραιη τιμή κατά ένα A ++ θα δώσει 11.

-- Ο τελεστής μειώνει μειώνει τον ακέραιο αριθμό κατά ένα A- θα δώσει 9.

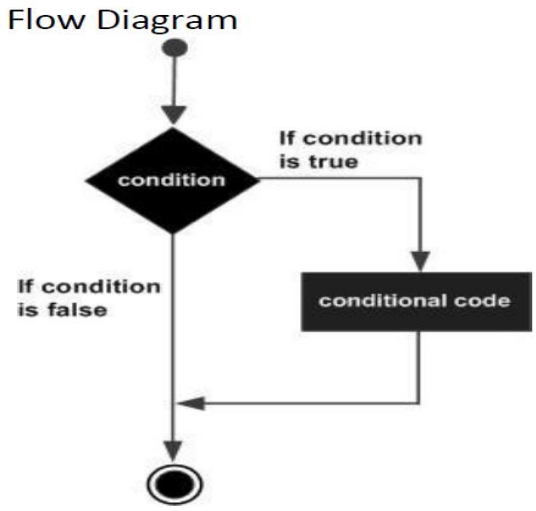
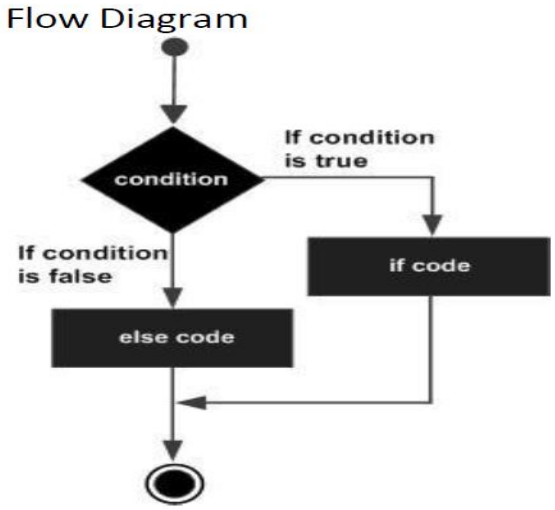
2.5 Δομές ελέγχου

Οι δομές ελέγχου που χρησιμοποιούμε προκειμένου να πάρουμε αποφάσεις και να ελέγξουμε την ροή τον εντολών μας είναι:

2.5.1 Η εντολή if

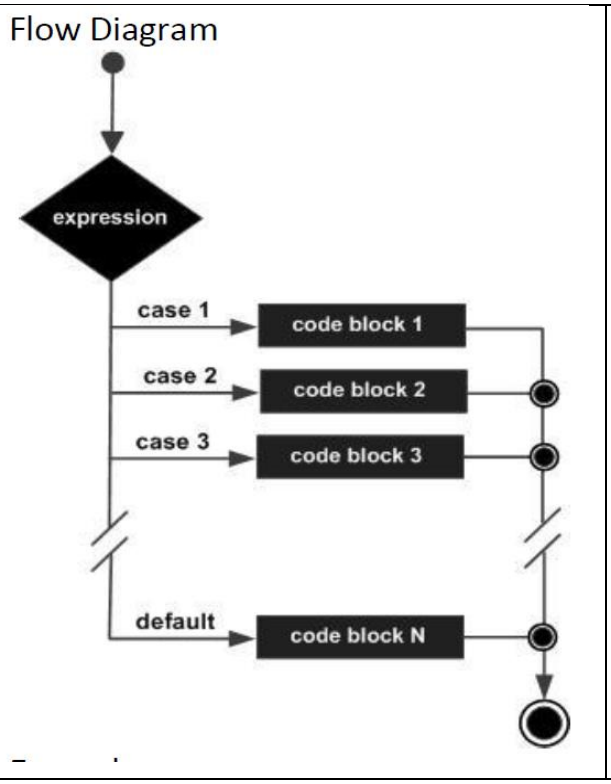
Η εντολή if χρησιμοποιείται για να ελεγχθεί μια συνθήκη. Αν η συνθήκη που ακολουθεί είναι αληθής, τότε εκτελείται ένα σύνολο εντολών που

ονομάζεται if-block, διαφορετικά πάμε στην εντολή else if όπου ελέγχετε η εκεί συνθήκη και αν είναι αληθής, αν ισχύει, τότε εκτελείται το σύνολο των εντολών που ονομάζεται else if -block, τέλος στην εντολή else εκτελείτε ένα άλλο σύνολο εντολών που ονομάζεται else-block. Η χρήση των όρων else if και else είναι προαιρετική.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ	ΠΑΡΑΔΕΙΓΜΑ ΕΝΤΟΛΩΝ
	`/* check the boolean condition using if` `if( a < 20 )` `{` `  /* if condition is true then print` `  printf("a is less than 20\n" );` `}` `printf("value of a is : %d\n", a);`
	`/* check the boolean condition */` `if( a < 20 )` `{` `  /* if condition is true then print t` `  printf("a is less than 20\n" );` `}` `else` `{` `  /* if condition is false then print` `  printf("a is not less than 20\n" );` `}`

2.5.2 Η εντολή switch

Η εντολή switch επιτρέπει των έλεγχο μιας μεταβλητής για ισότητα έναντι μιας λίστας τιμών. Κάθε τιμή ονομάζεται περίπτωση και η μεταβλητή που ενεργοποιείται ελέγχεται για κάθε περίπτωση switch.

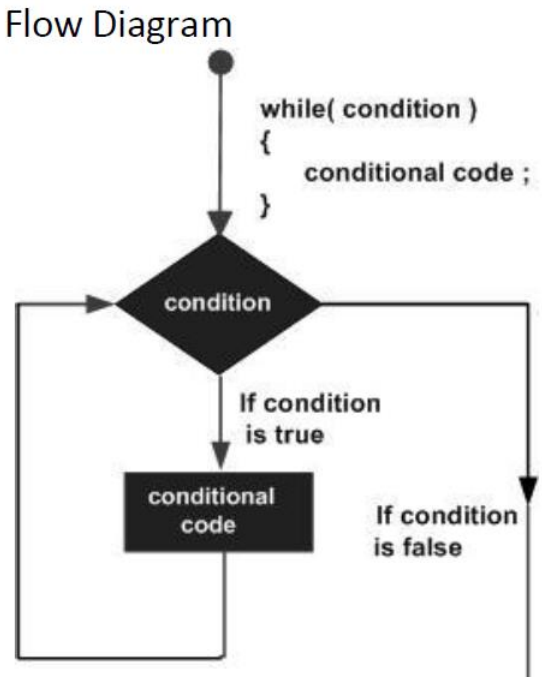
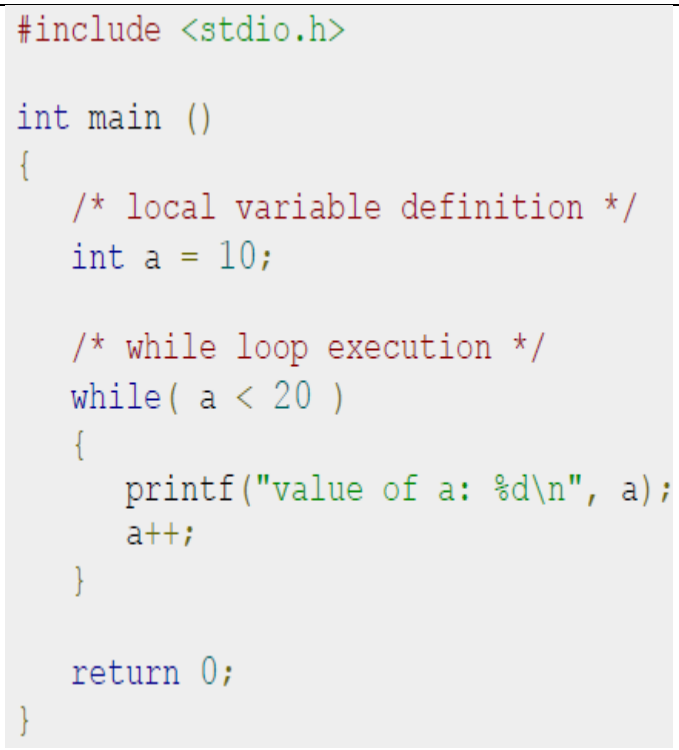
ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ	ΠΑΡΑΔΕΙΓΜΑ ΕΝΤΟΛΩΝ
	``` switch(expression){ case constant-expression : statement(s); break; /* optional */ case constant-expression : statement(s); break; /* optional */ /* you can have any number of case default : /* Optional */ statement(s); } ```

2.5.3 Η εντολή while

Αποτελεί δομή επανάληψης καθώς η εντολή while επιτρέπει να εκτελεστεί επανειλημμένα ένα σύνολο εντολών, όσο η λογική συνθήκη δίπλα στην εντολή while παραμένει αληθές. Όταν η λογική συνθήκη δίπλα στην εντολή while δεν είναι αληθής το σύνολο των εντολών δεν θα εκτελεστεί.

Εδώ, το βασικό σημείο του βρόχου while είναι ότι ο βρόχος μπορεί να μην εκτελεστεί ποτέ. Όταν η συνθήκη δοκιμάζεται και το αποτέλεσμα είναι ψευδές, το σώμα του βρόχου παραλείπεται και η πρώτη δήλωση μετά τον βρόχο while θα εκτελεστεί.

Επίσης χρησιμοποιείται στο πρόγραμμα μας η εντολή **while(1)** που η δυναμική της είναι να δημιουργεί έναν ατέρμον βρόχο, ο οποίοις όμως ελέγχεται στο εσωτερικό του για τερματισμό με την εντολή **break**.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ	ΠΑΡΑΔΕΙΓΜΑ ΕΝΤΟΛΩΝ
	


```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 )
   {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

2.5.4 Η εντολή for

Η δομή επανάληψης for είναι άλλη μία δομή επανάληψης, στην οποία επίσης υπάρχει επανειλημμένη εκτέλεση μιας ομάδας εντολών αλλά αυτή τη φορά σε προκαθορισμένο αριθμό επαναλήψεων. Σε αυτή την δομή επανάληψης, ορίζεται η αρχική τιμή και η τελική τιμή ενός δείκτη καθώς και το βήμα που θα αυξάνεται ή θα μειώνεται η τιμή του δείκτη αυτού.

Εδώ είναι η ροή του ελέγχου σε έναν βρόχο for:

1. Το βήμα init εκτελείται πρώτα και μόνο μία φορά. Αυτό το βήμα σας επιτρέπει να δηλώσετε και να αρχικοποιήσετε τυχόν μεταβλητές ελέγχου βρόχου.

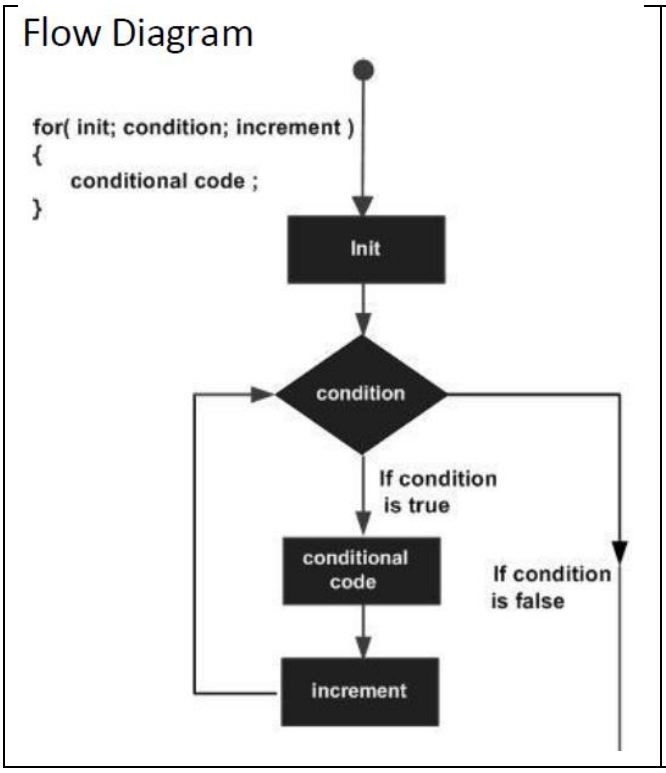
2. Στη συνέχεια, η κατάσταση αξιολογείται. Εάν είναι αλήθεια, το σώμα του βρόχου εκτελείται. Εάν είναι ψευδές, το σώμα του

βρόχου δεν εκτελείται και η ροή ελέγχου πηγαίνει στην επόμενη πρόταση αμέσως μετά τον βρόχο for.

3. Αφού εκτελεστεί το σώμα του βρόχου for, η ροή του στοιχείου ελέγχου πηγαίνει πίσω στη δήλωση αύξησης. Αυτή η δήλωση σάς επιτρέπει να ενημερώσετε τυχόν μεταβλητές ελέγχου βρόχου. Αυτή η πρόταση μπορεί να μείνει κενή, αρκεί να εμφανιστεί ένα ερωτηματικό μετά την συνθήκη.

4. Η κατάσταση τώρα αξιολογείται ξανά. Εάν είναι αληθές, ο βρόχος εκτελείται και η διαδικασία επαναλαμβάνεται (σώμα βρόχου, έπειτα βήμα αύξησης και στη συνέχεια πάλι συνθήκη). Αφού η συνθήκη γίνει ψευδής, ο βρόχος for τερματίζεται.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



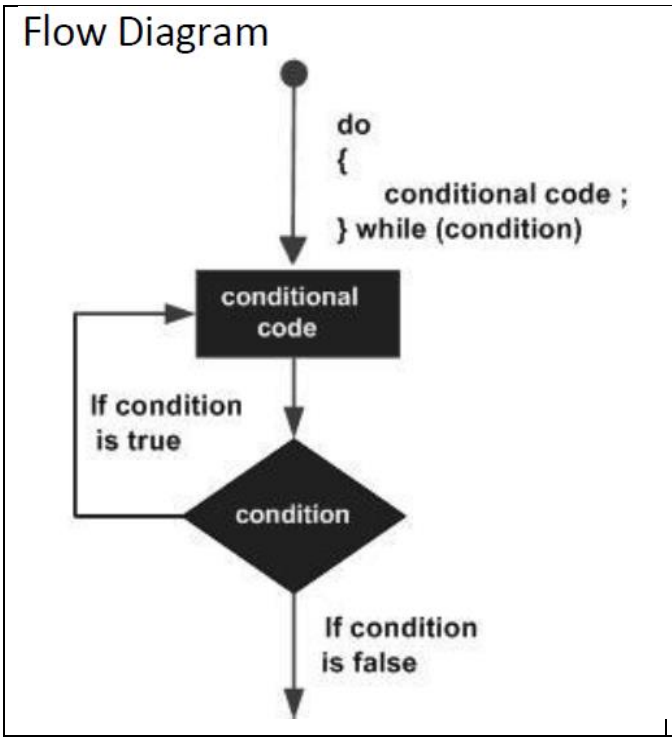
ΠΑΡΑΔΕΙΓΜΑ ΕΝΤΟΛΩΝ

```
/* for loop execution */
for( int a = 10; a < 20; a = a + 1 )
{
   printf("value of a: %d\n", a);
}
```

2.5.5 Η εντολή do while

Σε αντίθεση με τους βρόχους for και while, οι οποίοι δοκιμάζουν την κατάσταση – συνθήκη τερματισμού στην κορυφή του βρόχου, ο βρόχος do ... while στη γλώσσα προγραμματισμού C ελέγχει την συνθήκη του στο κάτω μέρος του βρόχου. Ο βρόχος do ... while είναι παρόμοιος με τον βρόχο while, εκτός από το ότι ο βρόχος do ... while είναι εγγυημένος ότι θα εκτελεστεί τουλάχιστον μία φορά.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



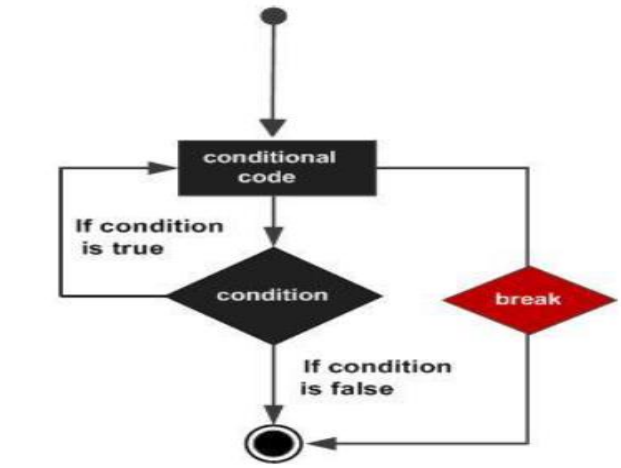
ΠΑΡΑΔΕΙΓΜΑ ΕΝΤΟΛΩΝ

```c
#include <stdio.h>

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
       printf("value of a: %d\n", a);
       a = a + 1;
   }while( a < 20 );

   return 0;
}
```

2.5.6 Η εντολή break

Η εντολή break χρησιμοποιείται για τη διακοπή μιας επαναλαμβανόμενης διαδικασίας εκτέλεσης εντολών, σε κάθε περίπτωση και ανεξάρτητα από το αν είναι αληθής ή ψευδής η συνθήκη στο βρόγχο επιλογής ή επανάληψης.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ **ΠΑΡΑΔΕΙΓΜΑ ΕΝΤΟΛΩΝ**

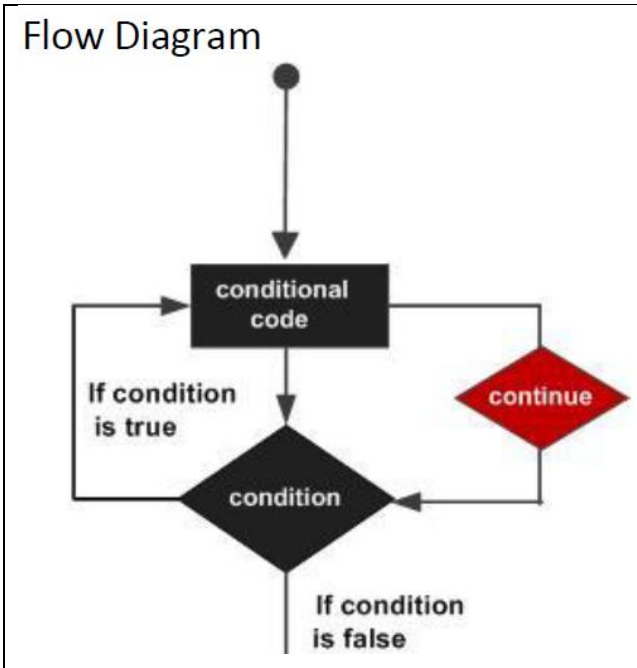



```
/* while loop execution */
while( a < 20 )
{
   printf("value of a: %d\n", a);
   a++;
   if( a > 15)
   {
      /* terminate the loop using break
       break;
   }
}
```

2.5.7 Η εντολή continue

Η εντολή continue χρησιμοποιείται για την παράληψη των υπολοίπων εντολών στο τμήμα των εντολών που εκτελούνται σε κάθε επανάληψη και να συνεχίσει με την επόμενη επανάληψη του βρόχου, **while(1)**

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ **ΠΑΡΑΔΕΙΓΜΑ ΕΝΤΟΛΩΝ**




```
/* do loop execution */
do
{
   if( a == 15)
   {
      /* skip the iteration */
      a = a + 1;
      continue;
   }
   printf("value of a: %d\n", a);
   a++;

}while( a < 20 );
```

2.6 Συναρτήσεις στην γλώσσα προγραμματισμού C

Η συνάρτηση αποτελεί ένα ξεχωριστό μέρος ενός προγράμματος. Έχει δικό της όνομα και παίρνει σαν είσοδο συγκεκριμένο αριθμό τιμών και επιστρέφει συγκεκριμένες τιμές. Ορίζει μια διαδικασία επεξεργασίας που μπορεί να χρησιμοποιηθεί πολλές φορές σε διάφορα σημεία το κώδικα.

Ο ορισμός και συγγραφή συνάρτησης στη γλώσσα προγραμματισμού C περιλαμβάνει ένα συνολο εντολών που αποτελείται από μια κεφαλίδα συνάρτησης και ένα σώμα συνάρτησης.

Ακολουθούν όλα τα μέρη μιας συνάρτησης:

Τύπος επιστροφής: Μια συνάρτηση μπορεί να επιστρέψει μια τιμή. Η εντολή return είναι ο τύπος δεδομένων της τιμής που επιστρέφει η συνάρτηση. Ορισμένες συναρτήσεις εκτελούν τις επιθυμητές λειτουργίες χωρίς να επιστρέψουν μια τιμή. Σε αυτήν την περίπτωση, η εντολή return είναι η λέξη - κλειδί κενό.

Όνομα συνάρτησης: Αυτό είναι το πραγματικό όνομα της συνάρτησης. Το όνομα της συνάρτησης και η λίστα παραμέτρων αποτελούν μαζί την υπογραφή της συνάρτησης.

Παράμετροι: Μια παράμετρος είναι σαν ένα σύμβολο κράτησης θέσης. Όταν καλείται μια συνάρτηση, περνάτε μια τιμή στην παράμετρο. Αυτή η τιμή αναφέρεται ως πραγματική παράμετρος ή όρισμα. Η λίστα παραμέτρων αναφέρεται στον τύπο, τη σειρά και τον αριθμό των παραμέτρων μιας συνάρτησης. Οι παράμετροι είναι προαιρετικές. Δηλαδή, μια συνάρτηση μπορεί να μην περιέχει παραμέτρους.

Σώμα συνάρτησης: Το σώμα συνάρτησης περιέχει μια συλλογή δηλώσεων που καθορίζουν τι κάνει η συνάρτηση.

Κατά τη δημιουργία μιας συνάρτησης σε γλώσσα προγραμματισμού C, καθορίζετε το τι πρέπει να κάνει η συνάρτηση. Για να χρησιμοποιήσετε μια συνάρτηση, θα πρέπει να την καλέσετε για να εκτελέσετε την καθορισμένη εργασία.

Όταν ένα πρόγραμμα καλεί μια συνάρτηση, ο έλεγχος προγράμματος μεταφέρεται στην κληθείσα συνάρτηση. Μια κληθείσα συνάρτηση εκτελεί καθορισμένη εργασία και όταν εκτελείται η δήλωση επιστροφής της, επιστρέφει τον έλεγχο προγράμματος πίσω στο κύριο πρόγραμμα. Για να καλέσετε μια συνάρτηση, απλά πρέπει να περάσετε τις απαιτούμενες παραμέτρους μαζί με το όνομα της συνάρτησης και εάν η συνάρτηση επιστρέψει μια τιμή, τότε μπορείτε να αποθηκεύσετε την επιστρεφόμενη τιμή.

Κλήση συνάρτησης με τιμή: Η μέθοδος κλήσης με τιμή σημαίνει ότι για τη μεταφορά τιμών χρησιμοποιούνται ορίσματα και η συνάρτηση αντιγράφει την πραγματική τιμή ενός ορίσματος στην τυπική παράμετρο της συνάρτησης. Σε αυτήν την περίπτωση, οι αλλαγές που γίνονται στην παράμετρο μέσα στη συνάρτηση δεν έχουν καμία επίδραση στο όρισμα. Από προεπιλογή, η γλώσσα προγραμματισμού C χρησιμοποιεί κλήση με μέθοδο τιμής για να περάσει ορίσματα. Γενικά, αυτό σημαίνει ότι ο κώδικας μέσα σε μια συνάρτηση δεν μπορεί να αλλάξει τα ορίσματα που χρησιμοποιούνται για την κλήση της συνάρτησης.

ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ

```c
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}
```

ΣΥΝΑΡΤΗΣΗ

```c
/* function returning the max between two
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Κλήση συνάρτησης με αναφορά: Η κλήση με μέθοδο αναφοράς για τη μεταφορά ορισμάτων σε μια λειτουργία αντιγράφει τη διεύθυνση ενός ορίσματος στην τυπική παράμετρο. Μέσα στη συνάρτηση, η διεύθυνση χρησιμοποιείται για πρόσβαση στο πραγματικό όρισμα που χρησιμοποιείται στην κλήση. Αυτό σημαίνει ότι οι αλλαγές που γίνονται στην παράμετρο επηρεάζουν το περασμένο όρισμα. Για να περάσετε την τιμή με αναφορά, οι δείκτες ορίσματος μεταφέρονται στις συναρτήσεις όπως κάθε άλλη τιμή. Συνεπώς, πρέπει να δηλώσετε τις παραμέτρους της συνάρτησης ως τύπους δείκτη όπως στην ακόλουθη εναλλαγή συνάρτησης (), η οποία ανταλλάσσει τις τιμές των δύο ακέραιων μεταβλητών που υποδεικνύονται από τα ορίσματά της.

ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ

```
int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values.
    * &a indicates pointer to a ie. address of variable
    * &b indicates pointer to b ie. address of variable
   */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );
```

ΣΥΝΑΡΤΗΣΗ

```
/* function definition to swap the v
void swap(int *x, int *y)
{
   int temp;
   temp = *x;     /* save the value a
   *x = *y;       /* put y into x */
   *y = temp;     /* put x into y */

   return;
}
```

2.7 Δομές Δεδομένων στην γλώσσα C

Οι δομές δεδομένων όπως λέει και το όνομά τους, είναι δομές που μπορούν να κρατήσουν μαζικά μερικά δεδομένα. Με άλλα λόγια χρησιμοποιούνται για να αποθηκεύουν δεδομένα που έχουν σχέση μεταξύ τους.

Αντί να δηλώσετε μεμονωμένες μεταβλητές, όπως αριθμός0, αριθμός1, ...αριθμός99, δηλώνετε μία μεταβλητή πίνακα όπως αριθμούς και χρησιμοποιούμε αριθμούς [0], αριθμούς [1] και ..., αριθμούς [99] για να αναπαραστήσετε μεμονωμένες μεταβλητές. Η πρόσβαση σε ένα συγκεκριμένο

στοιχείο σε έναν πίνακα γίνεται από ένα ευρετήριο. Όλοι οι πίνακες αποτελούνται από συνεχόμενες θέσεις μνήμης. Η χαμηλότερη διεύθυνση αντιστοιχεί στο πρώτο στοιχείο και η υψηλότερη διεύθυνση στο τελευταίο στοιχείο.

2.7.1 Μονοδιάστατοι Πίνακες

Για να δηλώσουμε έναν πίνακα στο C, καθορίζουμε πρώτα τον τύπο των και τον αριθμό των στοιχείων που απαιτούνται από έναν πίνακα ως εξής: **arrayName [arraySize];**

Το arraySize πρέπει να είναι μια ακέραια σταθερά μεγαλύτερη από μηδέν και ο τύπος μπορεί να είναι οποιοσδήποτε έγκυρος τύπος δεδομένων C. Για παράδειγμα, για να δηλώσετε έναν πίνακα 10 στοιχείων:**double balance[10];**

Τώρα το **balance** είναι μια μεταβλητή πίνακα που αρκεί για να κρατήσει έως και 10 πραγματικούς αριθμούς.

Η πρόσβαση σε ένα στοιχείο γίνεται με ευρετηρίαση του ονόματος του πίνακα. Αυτό γίνεται με την τοποθέτηση του ευρετηρίου του στοιχείου μέσα σε αγκύλες μετά το όνομα του πίνακα. Για παράδειγμα: **double salary = balance[9];**

Η παραπάνω δήλωση θα λάβει το 10ο στοιχείο από τον πίνακα και θα εκχωρήσει την τιμή στη μεταβλητή μισθού.

2.7.2 Δισδιάστατοι Πίνακες

Η απλούστερη μορφή του πολυδιάστατου πίνακα είναι ο δισδιάστατος πίνακας. Ένας δισδιάστατος πίνακας είναι, στην ουσία, ένας κατάλογος

μονοδιάστατων συστοιχιών. Για να δηλώσετε έναν δισδιάστατο ακέραιο πίνακα μεγέθους x, y θα γράφατε κάτι ως εξής: πληκτρολογήστε arrayName [x] [y];

Όπου ο τύπος μπορεί να είναι οποιοσδήποτε έγκυρος τύπος δεδομένων C και το arrayName θα είναι έγκυρο αναγνωριστικό C. Ένας δισδιάστατος πίνακας μπορεί να θεωρηθεί ως πίνακας που θα έχει x αριθμό γραμμών και y αριθμό στηλών. Ένας 2-διάστατος πίνακας a, ο οποίος περιέχει τρεις σειρές και τέσσερις στήλες μπορεί να εμφανιστεί ως εξής:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Εικόνα 2.1 Σχηματική αναπαράσταση δισδιάστατου πίνακα

Έτσι, κάθε στοιχείο στον πίνακα a αναγνωρίζεται από ένα όνομα στοιχείου της φόρμας a [i] [j], όπου a είναι το όνομα του πίνακα και i και j είναι οι συντελεστές που προσδιορίζουν μοναδικά κάθε στοιχείο στο a.

Η πρόσβαση σε ένα στοιχείο στον πίνακα δύο διαστάσεων γίνεται με τη χρήση των εγγραφών, δηλαδή του ευρετηρίου σειρών και του δείκτη στηλών του πίνακα. Για παράδειγμα: int val = a [2] [3]; Η παραπάνω δήλωση θα πάρει το 4ο στοιχείο από την 3η σειρά του πίνακα.

2.8 Συμπεράσματα

Η απλή σύνταξη της, η δυναμική χρήση της μνήμης, την καθιστούν την ιδανική γλώσσα απλών αλλά και σύνθετων εφαρμογών.

Ένα χαρακτηριστικό της ισχύος της C είναι πως έχουν αναπτυχθεί πολυάριθμες βιβλιοθήκες για κάθε είδους προγραμματιστική εργασία. Έτσι ο προγραμματιστής της γλώσσας, προκειμένου να κάνει μια συγκεκριμένη εργασία επιλέγει και εγκαθιστά και την αντίστοιχη βιβλιοθήκη, για την έκδοση της γλώσσας με την οποία εργάζεται. Μ' αυτόν τον τρόπο η γλώσσα αποκτά επεκτασιμότητα και γίνεται πιο φιλική για τον προγραμματιστή.

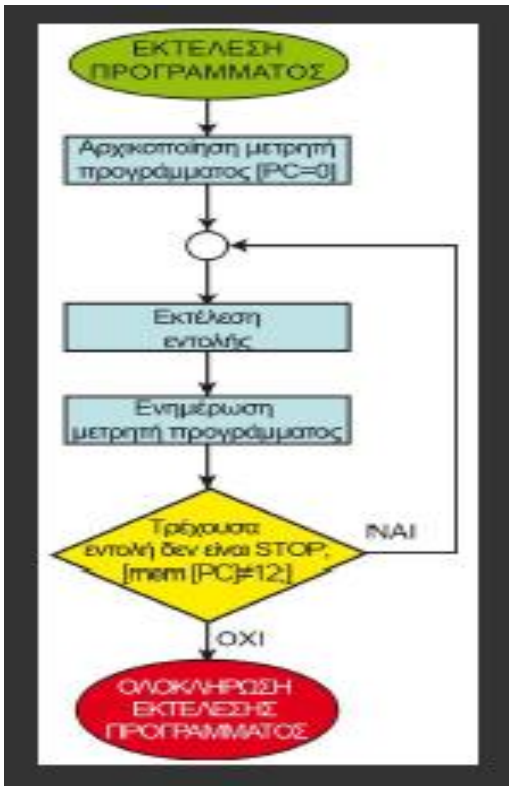
Κεφάλαιο 3ο

3.1 Εισαγωγή

Στο κεφάλαιο αυτό θα γίνει αναφορά στα βασικά συστατικά του προσομοιωτή που αναπτύξαμε και θα αναλύσουμε, τον βασικό τρόπο λειτουργίας του, την σύνδεση των δομικών του στοιχείων, την αλληλεπίδραση των διαφορετικών συστατικών μεταξύ τους. Επίσης θα γίνει παραλληλισμός των εργαλείων που χρησιμοποιήσαμε στην γλώσσα προγραμματισμού C για να καταφέρουμε να προγραμματίσουμε τις παραπάνω λειτουργίες.

3.2 Γενικό διάγραμμα λειτουργίας προσομοιωτή – μετρητής προγράμματος

Ο μετρητής προγράμματος (PC) δείχνει πάντα τη διεύθυνση της εντολής που πρόκειται να εκτελεστεί στον μικροεπεξεργαστή μας. Θεωρώντας ότι το πρόγραμμα ξεκινά πάντα από την τιμή 0, που είναι και η πρώτη θέση στην λίστα που χρησιμοποιούμε. Η εντολή που τερματίζει την εκτέλεση του προγράμματος είναι η STOP και όταν διαβαστεί σταματά και η εκτέλεση του προγράμματος. Προκειμένου να γίνει περισσότερο κατανοητή η λειτουργία του προσομοιωτή μας, παρατίθεται το διάγραμμα λειτουργίας του (Σχήμα 3.1) στο οποίο γίνεται αντιληπτή η γενικότερη λειτουργία του προγράμματος μας.



Σχήμα 3.1: Διάγραμμα λειτουργίας προσομοιωτή

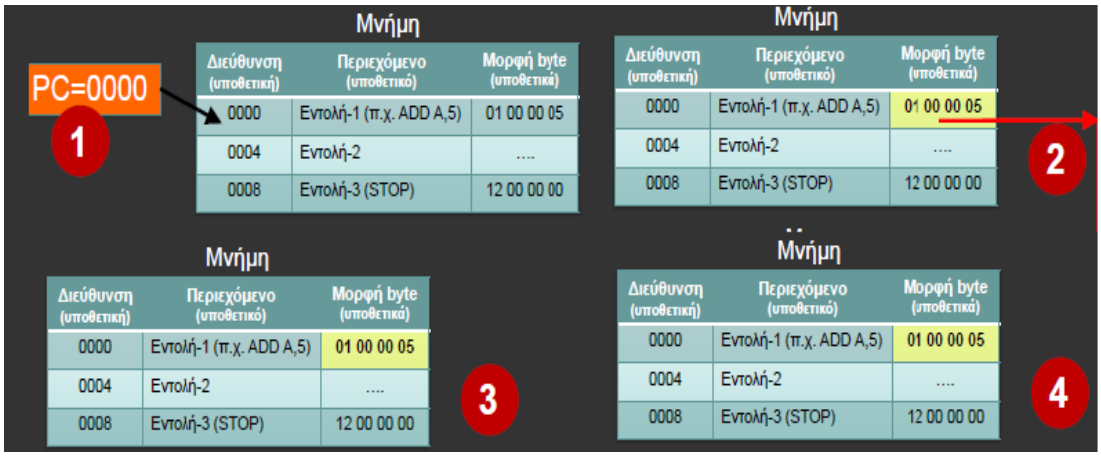
Να σημειώσουμε ότι ο μετρητής προγράμματος ενημερώνεται πριν την εκτέλεση της επόμενης εντολής, και κάθε εντολή αποθηκεύεται σ ένα μονοδιάστατο πίνακα με μέγεθος εντολής 4 bytes. Η σχετική αποθήκευση μιας εντολής στην μνήμη φαίνεται στο παρακάτω σχήμα (Σχήμα 3.2) και η διαδικασία εκτέλεσης μιας εντολής γίνεται σε τέσσερα στάδια:

Στάδιο 1: Έναρξη εκτέλεσης, θα εκτελεστεί η εντολή που «δείχνει» ο PC άρα και ενεργοποίηση διεύθυνσης 0000, δηλαδή της θέσης που βρίσκεται αυτή.

Στάδιο 2: Αντιγραφή, ο κώδικας της εντολής μαζί με τα ορίσματα, αντιγράφεται στο «σύστημα» της μονάδας ελέγχου.

Στάδιο 3: Το περιεχόμενο του καταχωρητή REG-A και το όρισμα (05) αντιγράφονται στην ΑΛΜ

Στάδιο 4: Εκτέλεση, γίνεται η πρόσθεση (REG-A + 05), έστω REG-A=2 και ενημερώνεται ο καταχωρητής κατάστασης (SREG). Επίσης αποθηκεύεται το αποτέλεσμα στον REG-A. Τα παραπάνω στάδια φαίνονται στο παρακάτω σχήμα (Σχήμα 3.2).



Σχήμα 3.2: Φόρτωση μιας εντολής στην μνήμη

Αυτό που πρέπει να αναφέρουμε και ισχύουν ως κανόνες-παραδοχές για την εύρωστη λειτουργία του προσομοιωτή είναι:

1. Κάθε εντολή αποτελείται από 4 byte.
2. Το πρώτο είναι ο μοναδικός κώδικας κάθε εντολής.
3. Το XX αντιπροσωπεύει το όρισμα (αν έχει η εντολή).

Η εκτέλεση του προγράμματος, όπως αναφέραμε και παραπάνω, ολοκληρώνεται (τερματισμός) όταν η ροή εκτέλεσης φτάσει στην εντολή STOP(κώδικας εντολής=12).

3.3 Εντολές που υποστηρίζονται από τον προσομοιωτή

Ο σκοπός του υποθετικού συστήματος είναι η εκτέλεση υποθετικών εντολών, επομένως θα εξηγήσουμε ποιες είναι οι υποστηριζόμενες εντολές και θα παραθέσουμε το ρεπερτόριο των εντολών αυτών, μαζί με τα ορίσματα με τα

οποία εφοδιάζονται. Οι υποθετικές εντολές εμφανίζονται στην παρακάτω εικόνα και παρατίθενται μαζί με την κωδικοποίηση τους.

Εντολή	Κωδικοποίηση	Λειτουργία
RD	01 00 00 00	(A ← KEYB) Ανάγνωση αριθμού από το πληκτρολόγιο και αποθήκευση στον καταχωρητή A
WR	02 00 00 00	(A → CON) Εμφάνιση περιεχομένων καταχωρητή A στην οθόνη
ADD B	03 00 00 00	(A ← A+B) Πρόσθεση περιεχομένων καταχωρητών A και B και αποθήκευση στον καταχωρητή A
MOV A,XX	04 00 00 XX	(A ← XX) Φόρτωση ακέραιου αριθμού στον καταχωρητή A
MOV B,XX	05 00 00 XX	(B ← XX) Φόρτωση ακέραιου αριθμού στον καταχωρητή B
INC A	06 00 00 00	(A ← A+1) Αύξηση του περιεχομένου του καταχωρητή A κατά 1
INC B	07 00 00 00	(B ← B+1) Αύξηση του περιεχομένου του καταχωρητή B κατά 1
DEC A	08 00 00 00	(A ← A-1) Μείωση του περιεχομένου του καταχωρητή A κατά 1
DEC B	09 00 00 00	(B ← B-1) Μείωση του περιεχομένου του καταχωρητή B κατά 1
JNZ A,XX	10 00 00 XX	(Αν A<>0, τότε μετάβαση στη διεύθυνση XX) Μετάβαση στη διεύθυνση XX αν το περιεχόμενο του καταχωρητή A δεν είναι μηδέν
JNZ B,XX	11 00 00 XX	(Αν B<>0, τότε μετάβαση στη διεύθυνση XX) Μετάβαση στη διεύθυνση XX αν το περιεχόμενο του καταχωρητή B δεν είναι μηδέν
STOP	12 00 00 00	(Exit) Τερματισμός της εκτέλεσης προγράμματος

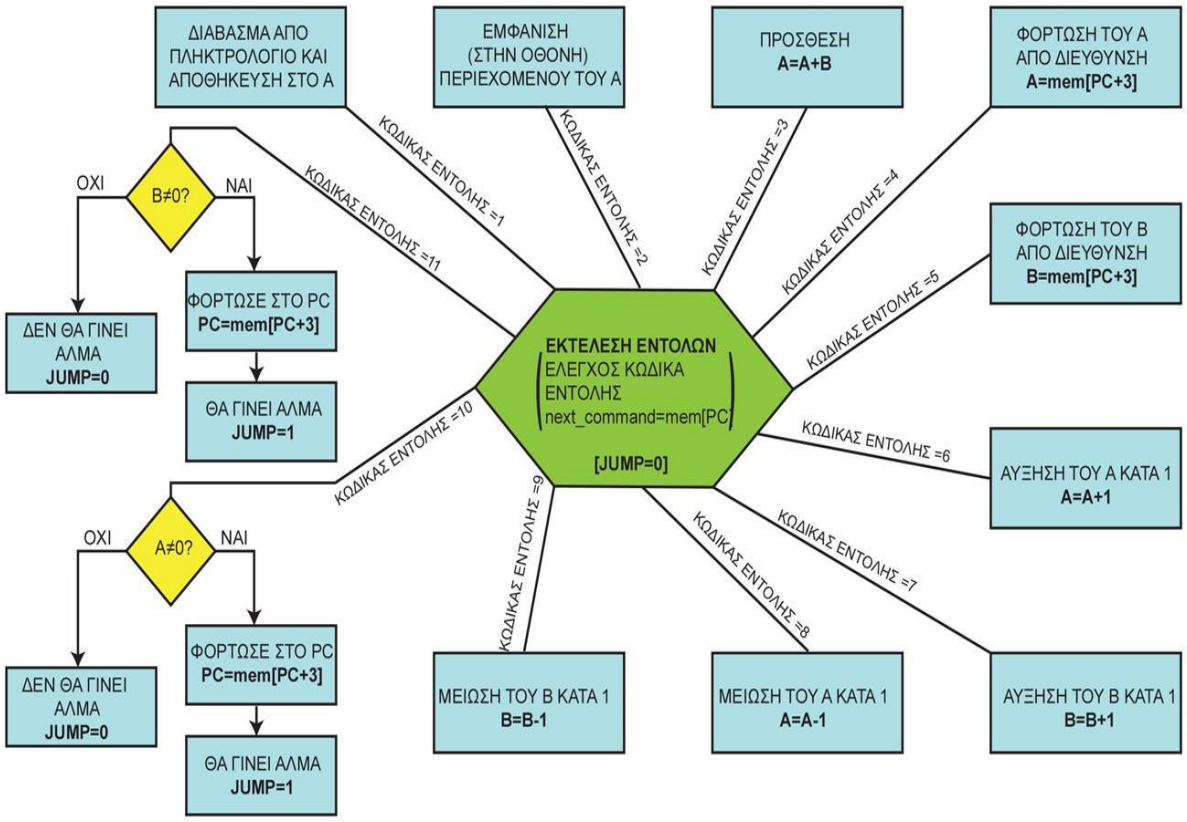
Εικόνα 3.1: Εντολές και ορίσματα προσομοιωτή

Στην εφαρμογή μας δεν υποστηρίζουμε καταχωρητή κατάστασης έτσι ώστε να ελέγχουμε περαιτέρω τις εντολές άλματος και ο έλεγχος μηδενισμού σε ένα άλμα υπό συνθήκη JNZ, γίνεται με έλεγχο του περιεχομένου του καταχωρητή που συνοδεύει την εντολή, και το όρισμα XX μας δείχνει την διεύθυνση μνήμης που περιέχει την εντολή, προς την οποία θα πραγματοποιηθεί το άλμα.

Έτσι κάθε φορά που πρόκειται να εκτελεστεί κάποια από τις παραπάνω εντολές, διαβάζουμε τον πρώτο από τους τέσσερις αριθμούς (κώδικας εντολής)

έτσι ώστε να καταλάβουμε για ποια εντολή πρόκειται και οι τρείς επόμενοι αριθμοί αφορούν τα ορίσματα που μπορεί να δέχεται η εντολή μας. Ουσιαστικά χρησιμοποιούμε μόνο τον τελευταίο αριθμό και τους υπόλοιπους τους αφήνουμε για μελλοντική επέκταση.

Χρησιμοποιούμε λοιπόν ένα πίνακα στον οποίο καταλαμβάνουμε για κάθε εντολή τέσσερις θέσεις. Η πρώτη θέση αφορά τον κώδικα εντολής και οι υπόλοιπες τα ορίσματα. Η επόμενη εντολή θα αποθηκευτεί μετά από τέσσερις θέσεις στον πίνακα και ο μετρητής προγράμματος θα μεταβάλλεται κατά τέσσερις θέσεις πάντα. Η προσπέλαση στον πίνακα με τις εντολές του υποθετικού προγράμματος θα είναι γραμμική και τα άλματα υπό συνθήκη θα μεταφέρουν στις αρχικές θέσεις για επανάληψη ή στις επόμενες θέσεις όταν κάνουμε δομή επιλογής. Στο παρακάτω σχήμα (Σχήμα 3.3) παρουσιάζεται ένα σενάριο όλων των πιθανών συνδυασμών που πρέπει να υποστηρίζει το υποθετικό μας σύστημα, ώστε να γίνει κατανοητή η λειτουργία του.



Σχήμα 3.3: Εκτέλεση υποθετικών εντολών

3.4 Βασικά συστατικά προσομοιωτή

Για την λειτουργία του υποθετικού προγράμματος θα χρησιμοποιήσουμε κάποιες μεταβλητές στις οποίες θα προσδώσουμε έναν ρόλο, σύμφωνα με τις απαιτήσεις του καταχωρητή μας. Πιο συγκεκριμένα θα χρησιμοποιήσουμε τους καταχωρητές – μεταβλητές A και B καθώς και την μεταβλητή PC ως μετρητή προγράμματος, όλες μεταβλητές τύπου ακεραίου. Για να αποθηκεύσουμε τις εντολές που υποστηρίζονται από το υποθετικό μας σύστημα, θα αρχικοποιήσουμε ένας ακόμη πίνακα στον οποίο θα κρατήσουμε τις εντολές αυτές σε μορφή χαρακτήρων.

Τέλος για την υποθετική μνήμη του συστήματος μας, θα δημιουργήσουμε ένα πίνακα 100 θέσεων, που θα αποτελέσει με τον μετρητή προγράμματος, την καρδιά του συστήματος μας. Ο παρακάτω πίνακας μας δείχνει τις μεταβλητές που πρόκειται να χρησιμοποιηθούν και τον ρόλο τους στο υποθετικό μας σύστημα.

Πίνακας 3.1: Βασικές μεταβλητές και πίνακες του προγράμματος

Όνομα Μεταβλητής	Τύπος	Ρόλος
PC	Ακέραιος	Μετρητής προγράμματος
A	Ακέραιος	Καταχωρητής A
B	Ακέραιος	Καταχωρητής B
mem	Χαρακτήρες	Μνήμη Συστήματος
ins	Ακέραιος	Αποθηκεμένες εντολές του προγράμματος που πρόκειται να εκτελεστεί

3.5 Βασικές συνιστώσες – υποπρογράμματα του προσομοιωτή

Το πρόγραμμα που έχει αναπτυχθεί αποτελείται από το κύριο μέρος του, στο οποίο πραγματοποιούνται όλες οι βασικές αρχικοποιήσεις και η εκτέλεση

των κατάλληλων υποπρογραμμάτων, με την σειρά που πρέπει, ώστε να έχουν νόημα οι εντολές που δίνονται προς εκτέλεση. Η διαχείριση των εντολών του προσομοιωτή γίνεται μέσα από ένα κεντρικό μενού επιλογών και ο ρόλος του κάθε υποπρογράμματος παρατίθεται στην παρακάτω εικόνα (Εικόνα 3.2).

Υποπρόγραμμα	Περιγραφή
menu	Εμφανίζει το κεντρικό μενού
command_list	Εμφανίζει στην οθόνη το όνομα (μνημονικό) κάθε διαθέσιμης εντολής μαζί με τον αντίστοιχο κώδικα
init_instructions	Αρχικοποιεί τον πίνακα με τις ονομασίες και τα ορίσματα των διαθέσιμων εντολών
execute	Πρόκειται για τον πυρήνα του συστήματος, αφού αναλαμβάνει την εκτέλεση των υποθετικών εντολών Assembly που έχει εισάγει ο χρήστης
print_mem_reg	Εμφανίζει στην οθόνη το περιεχόμενο των καταχωρητών A,B, του μετρητή προγράμματος (PC) καθώς και της υποθετικής μνήμης
enter_source	Διαδικασία εισαγωγής εντολών προγράμματος Assembly (με τον κώδικα εντολής) στην υποθετική μνήμη
help	Εμφανίζει βοηθητικό μήνυμα για το μενού επιλογών
clear_mem	Αρχικοποιεί με μηδενικά (καθαρισμός) τον πίνακα της υποθετικής μνήμης
empty_lines	Εμφανίζει κενές γραμμές για τον καθαρισμό περιοχής της οθόνης
read	Διαβάζει αλφαριθμητικό από το πληκτρολόγιο (αριθμό για το χρήστη) και επιστρέφει την αριθμητική του αξία

Εικόνα 3.2: Τα διαφορετικά τμήματα του προγράμματος

Η εκτέλεση των παραπάνω τμημάτων γίνεται με την εξής σειρά:

1. Εμφάνιση τίτλου εφαρμογής.
2. Εμφάνιση του μενού επιλογών που έχει ο χρήστης.
3. Ανάγνωση της επιλογής του χρήστη.
4. Κλήση του κατάλληλου υποπρογράμματος κατ' επιλογή του χρήστη.
5. Έξοδος από την λειτουργία του προγράμματος.

3.6 Επιπλέον δυνατότητες της εφαρμογής μας

Εκτός από τις παραπάνω διαθέσιμες λειτουργίες που θα μπορούσαμε να πούμε ότι αποτελούν τον πυρήνα της εφαρμογής μας, πρέπει να δίνεται η

δυνατότητα να μπορούν να εκτελεστούν κάποιες συμπληρωματικές λειτουργίες στον προσομοιωτή, ώστε να εξασφαλίζεται η χρηστικότητά του. Έτσι λοιπόν έχουν κατασκευαστεί κάποιες συμπληρωματικές λειτουργίες, που ελέγχονται από το μενού επιλογών και την επιλογή του χρήστη.

Αυτές οι συμπληρωματικές λειτουργίες φαίνονται στον παρακάτω πίνακα:

Πίνακας 3.2: Συμπληρωματικές λειτουργίες του προγράμματος

Επιλογή	Λειτουργία
Βοήθεια	Εμφάνιση κειμένου βοηθείας για τις επιλογές.
Εκτέλεση	Εκτέλεση προγράμματος που είναι αποθηκεμένο στην μνήμη, με τιμή εκκίνησης το 0.
Εισαγωγή κώδικα	Εισαγωγή κώδικα Assembly στην μνήμη χρησιμοποιώντας μενού επιλογών
Καθαρισμός Μνήμης	Αρχικοποίηση μνήμης - καθαρισμός
Λίστα Εντολών	Εμφάνιση των διαθέσιμων εντολών σε Assembly
Εμφάνιση μνήμης - καταχωρητών	Εμφάνιση περιεχομένων των καταχωρητών και της μνήμης
Έξοδος	Έξοδος από τον προσομοιωτή

3.7 Ανάπτυξη ψευδοκώδικα και επεξήγηση εντολών στην εφαρμογή μας

Στη ενότητα αυτή θα επεξηγήσουμε την βασική λειτουργία της κάθε εντολής υπο την μορφή ψευδοκώδικα, ώστε να γίνει καλύτερα αντιληπτή η βασική αρχή λειτουργίας της εφαρμογής μας, αλλά και η σύνδεση με την γλώσσα C που θα αναφερθούμε στο επόμενο κεφάλαιο. Έστω λοιπόν ο ακόλουθος ψευδοκώδικας κατά βήματα:

Βήμα 1: Θέσε τον μετρητή προγράμματος με 0

Βήμα 2: Επανάλαβε σε κάθε βήμα τις εντολές

Βήμα 3: στο άλμα θέσε τιμή 0

Βήμα 4: διάβασε την εντολή από την υποθετική μνήμη στη θέση που δείχνει ο μετρητής προγράμματος

Βήμα 5: Επίλεξε ποια εντολή έχει δοθεί και κάνε την κατάλληλη ενέργεια
Εάν έχει ζητηθεί άλμα θέσε στην μεταβλητή άλμα το 1

Βήμα 6: Εάν δεν έχει ζητηθεί άλμα αύξησε τον μετρητή προγράμματος κατά 4 διαφορετικά πήγαινε στην διεύθυνση του άλματος

Βήμα 7: Εάν έχει δοθεί η εντολή STOP σταμάτα διαφορετικά πήγαινε στο Βήμα 2

Ένα πραγματικό δοκιμαστικό πρόγραμμα, το οποίο θα δουλέψουμε στο επόμενο κεφάλαιο και θα δούμε την προσομοίωση του, παραθέτουμε στην επόμενη εικόνα (Εικόνα 3.3). Η σύγκριση με τον παραπάνω ψευδοκώδικα είναι πλέον εφικτή και η κατανόησή της λειτουργίας του πραγματοποιείται πιο εύκολα.

Κωδικοποίηση και λειτουργία

Κώδικας Assembly	Κωδικοποίηση 4 byte	Περιγραφή
MOV A,10	04 00 00 10	10 = όρισμα εντολής (A=10)
again:		
WR	02 00 00 00	Εμφάνιση περιεχομένου A στην οθόνη
DEC A	08 00 00 00	Μείωση του A κατά 1
JNZ A, again	10 00 00 again	Όσο A<>0 επαναφορά στη θέση again
STOP	12 00 00 00	Τερματισμός προγράμματος

Εικόνα 3.3: Δοκιμαστικό πρόγραμμα σε εντολές Assembly

3.8 Συμπεράσματα

Με την κατασκευή της εφαρμογής και την αντιμετώπιση των διαφόρων προβλημάτων που προκύπτουν, γίνεται πιο εύκολη η κατανόηση των εντολών συμβολικής γλώσσας, αλλά και του θεωρητικού μέρους του μαθήματος των μικροεπεξεργαστών. Η γλώσσες προγραμματισμού που κυριαρχούν στην εποχή μας, βρίσκονται πραγματικά πολύ κοντά στον άνθρωπο και αφήνουν στην μηχανή να κάνει όλη την κοπιαστική εργασία της αποκωδικοποίησης, εκσφαλμάτωσης και εκτέλεσης των προγραμμάτων που συντάσσουμε σ ένα σύγχρονο προγραμματιστικό περιβάλλον.

Κεφάλαιο 4ο

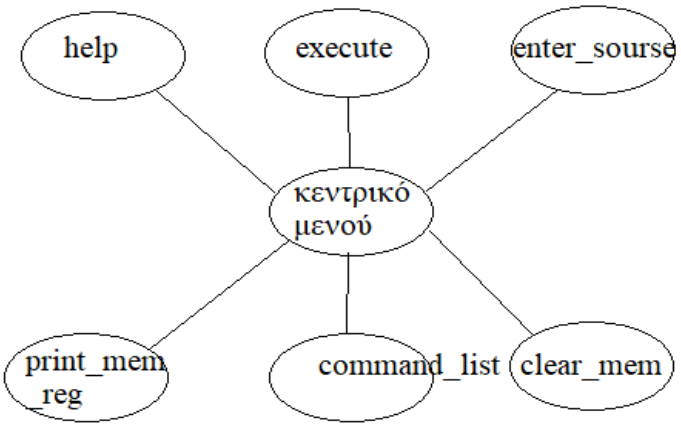
4.1 Εισαγωγή

Στο κεφάλαιο αυτό θα μελετήσουμε την ανάπτυξη των επιμέρους στοιχείων που χρησιμοποιήθηκαν, σε γλώσσα προγραμματισμού C και θα δείξουμε τον τρόπο με το οποίο συνδέθηκαν τα τμήματα αυτά μεταξύ τους. Δεν γίνεται αναφορά σε όλα τα συστατικά της εφαρμογής μας, παρά στα πιο καίρια, σύμφωνα πάντα με τις απόψεις του δημιουργού της εφαρμογής.

4.2 Ανάπτυξη του κυρίως προγράμματος

Με την έναρξη εκτέλεσης της εφαρμογής μας εμφανίζεται ένα κεντρικό μενού επιλογών και στην συνέχεια γίνεται η ανάγνωση δεδομένου από το πληκτρολόγιο, σύμφωνα με την επιλογή του χρήστη. Οι επιλογές που έχει ο χρήστης ενεργοποιούν αντίστοιχα υποπρογράμματα τα οποία εκτελούν ανάλογες εντολές και λειτουργίες. Στο παρακάτω σχήμα (Σχήμα 4.1) φαίνονται οι έξι επιλογές που μπορεί να επιλέξει ο χρήστης.

Στο τμήμα του κώδικα που παρατίθεται (Κώδικας 4.1) μπορούμε να μελετήσουμε τον τρόπο με τον οποίο κατασκευάζουμε τις βασικές εντολές της εφαρμογής μας. Στην αρχή των εντολών γίνονται οι αρχικοποιήσεις των μεταβλητών που μας χρειάζονται και στην συνέχεια με την κλήση των συναρτήσεων enter_source() και της συνάρτησης execute() ξεκινά η εφαρμογή μας.



Σχήμα 4.1: Κεντρικό μενού επιλογών εφαρμογής

Παρατηρήστε ότι με τις δύο εντολές, iinit_instructions() και command_list() γίνεται η αρχικοποίηση της υποθετικής μνήμης και των διαθέσιμων εντολών που δίνονται στον προσομοιωτή, καθώς χρησιμοποιούνται καθολικές μεταβλητές τύπου πίνακα, οι

```
char lst[12][100];
int mem[100];.

int main ()
{
     A=0;
     B=0;
     PC=0;
     system("chcp 1253");
     printf("\n Για εμφάνιση Ελληνικών χαρακτήρων!!\n");
     int choice;
     int flag = 0 ;
     printf ("\n\n Αρχικοποίηση Συστήματος\n\n!!");
     loading();
     init_instructions();
     command_list();
     init_mem();
     system("pause");
```

Κώδικας 4.1: Αρχικές εντολές της εφαρμογής

4.3 Το μενού επιλογών

Το υποπρόγραμμα αυτό εμφανίζει ένα μενού με διαθέσιμες επιλογές που μπορεί να επιλέξει ο χρήστης και επιτελεί την κατάλληλη λειτουργία. Παρατίθεται ο κώδικάς του, που δείχνει την λειτουργία του.

```
void menu(){

    printf ("\n--------[Main Menu]--------");
    printf ("\n[1] Help ");
    printf ("\n[2] Execute");
    printf ("\n[3] Insert Code ");
    printf ("\n[4] Init Memory");
    printf ("\n[5] Command List");
    printf ("\n[6] Show Registers Values");
    printf ("\n[0] Exit");
    printf ("\n-------------------------------");
    printf ("\nPlease give choice [1-6] or 0 for exit: ");
}
```

Κώδικας 4.2: Το μενού επιλογών της εφαρμογής

4.4 Εμφάνιση της λίστας εντολών

Προκειμένου να εμφανιστούν όλες οι διαθέσιμες εντολές με τις περιγραφές τους, που βρίσκονται αποθηκεμένες στον πίνακα lst, χρησιμοποιούμε το υποπρόγραμμα command_list(). Ουσιαστικά το υποπρόγραμμα προσπελαύνει τον πίνακα με δομή επανάληψης και τυπώνει τα περιεχόμενά του στην οθόνη.

```
void command_list(){
 int i;
    if (strcmp(lst[0], "")==0) {
        printf ("\n--------[ Λίστα Εντολών ]--------\n");
        printf ("- Η λίστα εντολών είναι κενή -\n");
    }else{
        printf ("\n--------[ Λίστα Εντολών ]--------\n");
        for (i = 0 ; i < 12 ; i++)
           printf("\n%s", lst[i]);
     printf("\n");
    }
}
```

Κώδικας 4.3: Εμφάνιση των εντολών της εφαρμογής

4.5 Αρχικοποίηση του πίνακα των εντολών

Με την βοήθεια του υποπρογράμματος init_instructions() πραγματοποιείται η αρχικοποίηση του πίνακα με τις εντολές και τις περιγραφές τους στον πίνακα ins. Πρόκειται για μία ανάθεση στατική, χωρίς την χρήση κάποιας δομής επανάληψης. Η συνάρτηση ουσιαστικά κατασκευάζει έναν πίνακα χαρακτήρων και τον επιστρέφει στο τμήμα του προγράμματος που κάλεσε το υποπρόγραμμα. Ο κώδικας της εφαρμογής φαίνεται στο παρακάτω τμήμα (Κώδικας 4.4).

```
void init_instructions(){

     sprintf(lst[0], "%s", "RD  ( A<-- KEYB  )     01 00 00 00"
);
     sprintf(lst[1], "%s", "WR  ( A--> CON   )     02 00 00 00"
);
     sprintf(lst[2], "%s", "ADD ( A <-- A+B  )     03 00 00 00"
);
     sprintf(lst[3], "%s", "MOV ( A <-- XX   )     04 00 00 00"
);
     sprintf(lst[4], "%s", "MOV ( B <-- XX   )     05 00 00 00"
);
     sprintf(lst[5], "%s", "INC ( A <-- A + 1)     06 00 00 00"
);
     sprintf(lst[6], "%s", "INC ( B <-- B + 1)     07 00 00 00"
);
     sprintf(lst[7], "%s", "DEC ( A <-- A - 1)     08 00 00 00"
);
     sprintf(lst[8], "%s", "DEC ( B <-- B - 1)     09 00 00 00"
);
     sprintf(lst[9], "%s", "JNC ( A <> 0 GOTO XX ) 10 00 00 00"
);
     sprintf(lst[10],"%s", "JNC ( b <> 0 GOTO XX ) 11 00 00 00"
);
     sprintf(lst[11],"%s", "STOP( Termination )    12 00 00 00"
);
}
```

Κώδικας 4.4: Αρχικοποίηση των εντολών της εφαρμογής

4.6 Κεντρική μονάδα επεξεργασίας

Αποτελεί τον πυρήνα της εφαρμογής μας καθώς εδώ παίρνουμε τις αποφάσεις για το ποια εντολή πρόκειται να εκτελεστεί, τι δεδομένα θα εμπλακούν και με πιο τρόπο θα συνδυαστούν οι εντολές που δίνει ο χρήστης. Η ιδέα είναι ότι ακολουθούμε τις εντολές που έχει δώσει ο χρήστης και είναι αποθηκευμένες στην υποθετική μνήμη ανά τέσσερις θέσεις. Εάν κάποια εντολή προσδιορίζει άλμα τότε η ροή εκτέλεσης θα μεταφερθεί στην θέση του πίνακα που δίνεται ως όρισμα στο άλμα (ετικέτα), διαφορετικά θα μεταφερθεί στην επόμενη εντολή.

Το παραπάνω σενάριο λειτουργίας επιτυγχάνετε με την χρήση ενός βρόχου, στο σώμα του οποίου γίνεται ο έλεγχος των εντολών προκειμένου να γίνει η κατάλληλή ενέργεια, λαμβάνοντας πάντα υπόψη τα εφόδια που πιθανόν θα πρέπει να ακολουθούν μια εντολή και την κατάλληλη θέση αυτών στον πίνακα των υποθετικών εντολών. Η επανάληψη σταματά όταν συναντήσουμε την εντολή 12 (εντολή STOP) που σηματοδοτεί το τέλος των εντολών. Τμήμα του κώδικα παρατίθεται παρακάτω.

```
void execute(){
     int jump;
     int next_command;
     PC = 0;
    A = 0;
    B = 0;
    printf ("\n Εκτέλεση Προγράμματος!!\n");

    while(1){
        jump=0;
        next_command = mem[PC];
        // check
        printf("\ncommand=%d PC = %d",next_command,PC);

        if (next_command == 0){
            printf ("\n\nGive value for A= ");
            A = read_kb();
        }
        else if (next_command == 1)
            printf("\n\nThe value of A = %d", A);
        else if (next_command == 2) {
            A=A+B ;
            printf("\n\nCommand 3 (A=A+B) = %d",A);
```

```
        }
        else if (next_command == 3) {
            A=mem[PC+3];
            printf("\n\nCommand 4 (load A) =%d",A);
        }
      else if (next_command == 4) {
            B=mem[PC+3];
            printf("\n\nCommand 5 (load B) =%d",B);
        }
        else if (next_command == 5) {
            A = A +1;
            printf("\n\nCommand 6 (inc A) = %d",A);
    }
        else if (next_command == 6) {
            B = B +1;
            printf("\n\nCommand 7 (inc B) =%d",B);
        }
          else if (next_command == 7) {
            A = A - 1 ;
            printf("\n\nCommand 8 (dec A) =%d",A);
        }
        else if (next_command == 8) {
            B = B - 1;
            printf("\n\nCommand 9 (dec B) =%d",B);
        }
        else if (next_command == 9) {
            if ( A != 0 ) {
                PC = mem[PC+3] ;
                jump=1;
            }
            else
                jump=0;
        }
      else if (next_command == 10) {
            if (B != 0 ) {
                PC = mem[PC+3];
                jump=1;
            }
            else
                jump=0;
        }
        else if (next_command == 11)
            break;

        if (jump == 0 ) // βήμα στην επανάληψη!!
            PC=PC+4;
    }
   printf("\n\n Το πρόγραμμα ολοκληρώθηκε!!\n\n");
}
```

Κώδικας 4.5: Η CPU της εφαρμογής μας

4.7 Εισαγωγή εντολών στην υποθετική μνήμη

Με το συγκεκριμένο υποπρόγραμμα πραγματοποιείται η εισαγωγή των εντολών που πρόκειται να εκτελεστούν στην υποθετική μνήμη. Χρησιμοποιείται δομή επανάληψης και εδώ μέχρι να δοθεί η εντολή 12 που σηματοδοτεί το τέλος των εντολών.

Το σημείο που πρέπει να προσεχθεί είναι ότι κατά την εισαγωγή των εντολών πρέπει να γίνει διαλογή ώστε εάν κάποια από τις εντολές που εισάγετε απαιτεί όρισμα, αυτό να ζητείται κατάλληλα από τον χρήστη, και να τοποθετείται στο κατάλληλο σημείο της υποθετικής μνήμης. Το καίριο τμήμα του υποπρογράμματος φαίνεται στον παρακάτω τμήμα κώδικα.

```
void enter_source(){

    printf ("\n--------[Εισαγωγή κώδικα εντολής]--------");
    printf ("----------------\n");
    int args = 0;
    PC =  0;
    int command = 0;
    int XX = 0;

    while (1){
         print_mem_reg(lst,mem);
        printf ("\nΑρχική Διεύθυνση ['%d']:", PC);
        command=read_kb();

        printf ("\n Check 1 dwsate tin entoli : %s\n",
lst[command-1]);
        mem[PC]=(command-1);
        if (command == 4){
            printf("\nH εντολή 4( MOV A, XX ) απαιτεί
όρισμα [XX]=");
            XX=read_kb();
            mem[PC+1]=(0);
            mem[PC+2]=(0);
            mem[PC+3]=(XX);
            A=XX;
        }
        else if (command == 5){
            printf("\nH εντολή 5( MOV B, XX ) απαιτεί
όρισμα [XX]=");
            XX=read_kb();
            mem[PC+1]=(0);
```

```
            mem[PC+2]=(0);
            mem[PC+3]=(XX);
            B=XX;
        }
        else if (command == 10){
             printf("\nH εντολή 10( JNZ A, XX ) απαιτεί
όρισμα [XX]=");
            XX=read_kb();
            mem[PC+1]=(0);
            mem[PC+2]=(0);
            mem[PC+3]=(XX);
        }
        else if (command == 11){
          printf("\nH εντολή 11( JNZ B, XX ) απαιτεί όρισμα
[XX]=");
            XX=read_kb();
            mem[PC+1]=(0);
            mem[PC+2]=(0);
            mem[PC+3]=(XX);
        }
        else if (command==3){
            A=A+B;
        }
        PC = PC + 4 ;
        if(command == 12 )
            break;
    }
    print_mem_reg(lst,mem);
}
```

Κώδικας 4.6:Δημιουργία υποθετικής μνήμης και ενημέρωση της

4.8 Εμφάνιση βοηθητικού κειμένου

Με το συγκεκριμένο υποπρόγραμμα πραγματοποιείται η εισαγωγή των εντολών που πρόκειται να εκτελεστούν στην υποθετική μνήμη. Χρησιμοποιείται δομή επανάληψης και εδώ μέχρι να δοθεί η εντολή 12 που σηματοδοτεί το τέλος των εντολών.

Το σημείο που πρέπει να προσεχθεί είναι ότι κατά την εισαγωγή των εντολών πρέπει να γίνει διαλογή ώστε εάν κάποια από τις εντολές που εισάγετε

απαιτεί όρισμα, αυτό να ζητείται κατάλληλα από τον χρήστη, και να τοποθετείται στο κατάλληλο σημείο της υποθετικής μνήμης. Το καίριο τμήμα του υποπρογράμματος φαίνεται στον παρακάτω τμήμα κώδικα.

```
void help(){

    printf ("\n--------[Κείμενο Βοήθειας]--------\n");
    printf ("[1] Help - Βοήθεια = Εμφάνιση του κειμένου που
διαβάζετε τώρα!!\n");
    printf ("[2] Εκτέλεση = Εκτέλεση προγράμματος Assembly
ξεκινώντας απο την διέυθυνση 0 (Μετρητής Προγράμματος
=0)\n");
    printf ("[3] Εισαγωγή Κώδικα = Εισαγωγή κώδικα Assembly
στην μνήμη χρησιμοποιώντας μενού εντολών\n");
    printf ("[4] Καθαρισμός Μνήμης = Αρχικοποίηση μνήμης
(θέτει όλα τα περιεχόμενα ως μηδέν)\n");
    printf ("[5] Λίστα Εντολών = Εμφάνιση των διαθέσιμων
εντολών Assembly με τους αντίστοιχους κώδικές\n");
    printf ("[6] Εμφάνιση Μνήμης/Καταχωρητών = Εμφάνιση
περιεχομένων μνήμης και καταχωρητών\n");
    printf ("[0] Εξοδος = Έξοδος απο τον προσομοιωτή\n");
    printf ("-------------------------------------\n\n");
 }
```

Κώδικας 4.7: Η επιλογή βοήθειας της εφαρμογής μας

4.9 Συμπεράσματα

Μια γλώσσα προγραμματισμού υψηλού επιπέδου, όπως είναι η γλώσσα προγραμματισμού C, δίνει στον προγραμματιστή πάρα πολλές δυνατότητες προκειμένου να σχεδιάσει, να εκπονήσει και τέλος να αποσφαλματώσει εφαρμογές με πολύ μεγάλη ευκολία, σύντομα και με τρόπο κατανοητό στον αρχάριο προγραμματιστή.

Κεφάλαιο 5ο

5.1 Εισαγωγή

Αυτή η ενότητα περιγράφει τον τρόπο ρύθμισης του περιβάλλοντος του συστήματός σας πριν αρχίσετε να πραγματοποιείται κάποιου είδους προγραμματισμού χρησιμοποιώντας γλώσσα C.

Πριν ξεκινήσετε να προγραμματίζετε χρησιμοποιώντας τη γλώσσα προγραμματισμού C, χρειάζεστε τα ακόλουθα δύο λογισμικά διαθέσιμα στον υπολογιστή σας,

(α) Text Editor και

(β) The C Compiler.

5.2 Επεξεργαστής κειμένου

Αυτό το λογισμικό θα χρησιμοποιηθεί για να πληκτρολογήσετε το πηγαίο σας πρόγραμμά σας. Παραδείγματα τέτοιου λογισμικού είναι το σημειωματάριο των Windows, το Brief, το Epsilon, το EMACS και το vim ή vi. Το όνομα και η έκδοση του επεξεργαστή κειμένου μπορεί να διαφέρουν σε διαφορετικά λειτουργικά συστήματα. Για παράδειγμα, το Σημειωματάριο θα χρησιμοποιηθεί σε Windows και το vim ή vi μπορεί να χρησιμοποιηθεί σε παράθυρα καθώς και Linux ή UNIX.

Τα αρχεία που δημιουργείτε με τον επεξεργαστή σας ονομάζονται αρχεία προέλευσης και περιέχουν πηγαίο κώδικα προγράμματος. Τα αρχεία προέλευσης για προγράμματα C συνήθως ονομάζονται με την επέκταση ".c".

Πριν ξεκινήσετε τον προγραμματισμό σας, βεβαιωθείτε ότι έχετε έναν επεξεργαστή κειμένου και ότι έχετε αρκετή εμπειρία για να γράψετε ένα πρόγραμμα υπολογιστή, να το αποθηκεύσετε σε ένα αρχείο, να το μεταγλωττίσετε και τελικά να το εκτελέσετε.

5.3 Ο C Compiler

Ο πηγαίος κώδικας που γράφεται στο αρχείο προέλευσης είναι η αναγνώσιμη από τον άνθρωπο πηγή για το πρόγραμμά σας. Πρέπει να "μεταγλωττιστεί", να μετατραπεί σε γλώσσα μηχανής, έτσι ώστε η CPU σας να μπορεί να εκτελέσει το πρόγραμμα σύμφωνα με τις οδηγίες που δίνονται. Αυτός ο μεταγλωττιστής γλώσσας προγραμματισμού C θα χρησιμοποιηθεί για τη μεταγλώττιση του πηγαίου κώδικα στο τελικό εκτελέσιμο πρόγραμμα.

Ο πιο συχνά χρησιμοποιούμενος και δωρεάν διαθέσιμος μεταγλωττιστής είναι ο μεταγλωττιστής GNU C/C ++, διαφορετικά μπορείτε να έχετε μεταγλωττιστές είτε από την HP είτε από την Solaris εάν διαθέτετε αντίστοιχα λειτουργικά συστήματα.

Η παρακάτω ενότητα σας καθοδηγεί για τον τρόπο εγκατάστασης του μεταγλωττιστή GNU C/C ++ σε διάφορα λειτουργικά συστήματα. Αναφέρω C/C ++ μαζί επειδή ο μεταγλωττιστής GNU gcc λειτουργεί και για γλώσσες προγραμματισμού C και C ++.

5.4 Εγκατάσταση σε UNIX /Linux

Εάν χρησιμοποιείτε Linux ή UNIX, ελέγξτε αν το GCC είναι εγκατεστημένο στο σύστημά σας εισάγοντας την ακόλουθη εντολή από τη γραμμή εντολών: $ gcc -v Εάν έχετε εγκαταστήσει μεταγλωττιστή GNU στο μηχάνημά σας, τότε θα πρέπει να εκτυπώσει ένα μήνυμα ως εξής :

```
Using built-in specs.
Target: i386-redhat-linux
Configured with: ../configure --prefix=/usr .......
Thread model: posix
gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)
```

Εάν το GCC δεν είναι εγκατεστημένο, τότε θα πρέπει να το εγκαταστήσετε μόνοι σας χρησιμοποιώντας τις αναλυτικές οδηγίες που είναι διαθέσιμες στη διεύθυνση: *http://gcc.gnu.org/install/*

5.5 Εγκατάσταση σε Mac OS

Εάν χρησιμοποιείτε Mac OS X, ο ευκολότερος τρόπος για να αποκτήσετε το GCC είναι να κατεβάσετε το περιβάλλον ανάπτυξης Xcode από τον ιστότοπο της Apple και να ακολουθήσετε τις απλές οδηγίες εγκατάστασης. Μόλις ρυθμίσετε το Xcode, θα μπορείτε να χρησιμοποιήσετε μεταγλωττιστή GNU για C/C ++. Το Xcode είναι προς το παρόν διαθέσιμο στη διεύθυνση developer.apple.com/technologies/tools/.

5.6 Εγκατάσταση σε Windows

Για να εγκαταστήσετε το GCC στα Windows, πρέπει να εγκαταστήσετε το MinGW. Για να εγκαταστήσετε το MinGW, μεταβείτε στην αρχική σελίδα του MinGW, www.mingw.org και ακολουθήστε τον σύνδεσμο στη σελίδα λήψης του MinGW. Κατεβάστε την τελευταία έκδοση του προγράμματος εγκατάστασης MinGW, η οποία θα πρέπει να ονομάζεται MinGW- <version> .exe. Κατά την εγκατάσταση του MinWG, τουλάχιστον, πρέπει να εγκαταστήσετε gcc-core, gcc-g ++, binutils και τον χρόνο εκτέλεσης του MinGW, αλλά μπορεί να θέλετε να εγκαταστήσετε περισσότερα.

Προσθέστε τον υποκατάλογο κάδου της εγκατάστασης MinGW στη μεταβλητή περιβάλλοντος PATH, ώστε να μπορείτε να καθορίσετε αυτά τα εργαλεία στη γραμμή εντολών με τα απλά ονόματά τους. Όταν ολοκληρωθεί η εγκατάσταση, θα μπορείτε να εκτελέσετε gcc, g ++, ar, ranlib, dlltool και πολλά άλλα εργαλεία GNU από τη γραμμή εντολών των Windows.

Κεφάλαιο 6ο

Συμπεράσματα και μελλοντική εργασία

Σ’ αυτή τη εργασία στα πλαίσια της διδακτορικής μου μελέτης, έγινε αναφορά και χρησιμοποιήθηκαν έννοιες για την εφαρμογή αλγοριθμικών εργαλείων: (1) στον ενεργειακά αποδοτικό προγραμματισμό εργασιών σε επεξεργαστές κατανεμημένων / cloud με δυνατότητα DVFS. και (2) πρόβλεψη και μοντελοποίηση των επιδόσεων τους. Πιο συγκεκριμένα μελετήθηκε ο ενεργειακώς αποδοτικός προγραμματισμός των εργασιών σε επεξεργαστές με δυνατότητα DVFS και δόθηκε μια εις βάθος έρευνα των υπαρχουσών μεθόδων.

Βασικός σκοπός της εργασίας ήταν η ανάπτυξη της κατάλληλης εφαρμογής και η χρήση των εργαλείων που παρέχονται από αυτήν, ώστε να γίνει μια όσο το δυνατόν καλύτερη προσέγγιση ενός προσομοιωτή εντολών assembly. Η επιλογή της γλώσσας προγραμματισμού ήταν η κατάλληλη, καθώς μας παρείχε όλα τα απαραίτητα εργαλεία για την κατασκευή μιας τέτοιας εφαρμογής. Οι γλώσσες προγραμματισμού υψηλού επιπέδου, γενικότερα οι γλώσσες που είναι κοντά στον άνθρωπο, παρέχουν κατανόηση στην χρήση των δομικών τους στοιχείων, μεγαλύτερη γκάμα επιλογών και εργαλείων και γενικότερα κάνουν αυτό για το οποίο είναι σχεδιασμένες, να βοηθούν στην ευκολότερη ανάπτυξη προγραμμάτων, με εποπτικότητα και κατανόηση του κώδικα, όχι μόνο από την μεριά του δημιουργού, άλλα και από την μεριά του αναγνώστη- παρατηρητή.

Χρησιμοποιώντας ως βάση αυτή την εφαρμογή μπορούμε να αναπτύξουμε έναν προσομοιωτή με περισσότερες δυνατότητες που θα μπορούσαν να είναι:

• Αντικατάσταση αποθηκευμένης εντολής στην μνήμη ή τροποποίηση μιας ήδη υπάρχουσας εντολής.

• Εκτέλεση προγράμματος βήμα προς βήμα και εμφάνιση μηνυμάτων.

• Φόρτωση των εντολών από αρχείο και εκτέλεση τους.

• Παρακολούθηση περιεχομένου των καταχωρητών κατά την διάρκεια της εκτέλεσης του κώδικα.

Επίσης θα μπορούσαμε να υλοποιήσουμε ξεχωριστά την κεντρική μονάδα επεξεργασίας από την αριθμητική και λογική μονάδα, να προσθέσουμε καταχωρητή εντολών, ώστε η εφαρμογή μας να πλησιάζει περισσότερο την κλασσική δομή ενός μικροεπεξεργαστή.

Βιβλιογραφία

Ελληνόφωνη

- Κατσένος, Πέτρος, 2020. Ανάπτυξη λογισμικού γεωστατιστικών μεθόδων χωρικής παρεμβολής σε περιβάλλον G.I.S. Μεταπτυχιακή διατριβή. Γεωπονικό Πανεπιστήμιο Αθήνας.

- Παπάζογλου, Παναγιώτης, 2021. Προχωρημένα θέματα προγραμματισμού σε συμβολική γλώσσα. Εργαστηριακές σημειώσεις αρχιτεκτονικής υπολογιστών. Τεχνολογικό ίδρυμα Λαμίας.

- Κωνσταντίνα Κατσάνου, 2020, Ανάπτυξη προσομοιωτή σε γλώσσα προγραμματισμού Python, Πτυχιακή Εργασία, Τεχνολογικό ίδρυμα Λαμίας.

Ξενόγλωσση

- Kirk W. Cameron, C PROGRAMMING TUTORIAL Simply Easy Learning, 2021.

- Lokman kesen, IMPLEMENTATION OF AN 8-BIT MICROCONTROLLER WITH SYSTEM C*,* PhD Thesis, Auburn University, 2021.

- Tim Bailey, An Introduction to the C Programming Languageand Software Design, 2021

- Systems, Dublin, Ireland, 2020.