Τεχνολογικό Πανεπιστήμιο Κύπρου
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ και Πληροφορικής
Φθινοπωρινό Εξάμηνο 2013 - 2014
ΜΗΥΠ 323: Αρχιτεκτονικη Η/Υ
Μιχάλης Προδρόμου – Μαριαλένα Σιαμμά
Τελική Εργασία – Αναφορά 1ου Milestone

Εισαγωγή:

Ο κώδικας του προγράμματός μας αποτελείται συνολικά από 5 διαφορετικά κομμάτια. Υπάρχει το source.c και τέσσερα διαφορετικά header files, η λειτουργία των οποίων περιγράφεται πιο κάτω. Ο λόγος που χωρίσαμε με αυτόν τον τρόπο τον κώδικά μας είναι για να πετύχουμε καλύτερη σύνταξη, πιο εύκολη οργάνωση και να είναι το πρόγραμμα όσο το δυνατόν πιο ευκολοκατανόητο. Με αυτό τον διαχωρισμό, ήταν πιο εύκολη και η υλοποίηση κάποιων ανεξάρτητων κομματιών του κώδικα, τα οποία έπρεπε να γράψει ή να αλλάξει μόνο το ένα άτομο από την ομάδα, στις περιπτώσεις όπου δεν ήταν δυνατό να γίνει κάποια συνάντηση μεταξύ των ατόμων που την αποτελούν.

Περιγραφή λειτουργίας προγράμματος:

1. Αρχικά ο χρήστης θα πρέπει να περάσει όλες τις τιμές των παραμέτρων του προγράμματος στο αρχείο *Libraries_and_defines.h*. Βάση αυτών των παραμέτρων, θα γίνονται οι υπολογισμοί για τα πεδία των tag, index και block offset σε μετέπειτα στάδιο. Όλα τα υπόλοιπα header files αλλά και το source.c, συμπεριλαμβάνουν στην αρχή του κώδικά τους, τους ορισμούς και αρχικοποιήσεις των παραμέτρων που ανήκουν στο αρχείο αυτό, έτσι ώστε να μπορούν να χρησιμοποιούν αυτές τις τιμές. Επίσης σε αυτό το header file, περιλαμβάνονται και όλες οι βιβλιοθήκες της γλώσσας C τις οποίες χρησιμοποιούμε στο πρόγραμμα μας, ώστε να μπορούμε να χρησιμοποιούμε κάποιες έτοιμες συναρτήσεις που μας προσφέρει η συγκεκριμένη γλώσσα.
2. Αφού δηλωθούν οι παράμετροι από τον χρήστη, μπορούμε να εκτελέσουμε το κυρίως πρόγραμμα το οποίο βρίσκεται στο αρχείο *Source.c* και περιλαμβάνει την συνάρτηση *main().* Εδώ αρχικά κάνουμε include το αρχείο *Libraries_and_defines.h* καθώς και όλα τα υπόλοιπα header files , έτσι ώστε να μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις που περιέχονται σε αυτά τα αρχεία, στην συνέχεια του προγράμματος. Ακολουθούν διάφορες δηλώσεις μεταβλητών τις οποίες χρησιμοποιεί το πρόγραμμα, καθώς και το κάλεσμα της συνάρτησης *structures(),* η οποία βρίσκεται στο αρχείο structures.h.
3. *Structures.h :* Σε αυτό το αρχείο γίνονται όλοι οι μαθηματικοί υπολογισμοί που χρειάζονται για την εύρεση των αριθμών των ψηφίων που απαιτούνται για το κάθε πεδίο (πχ: πόσα ψηφία χρειάζονται για την αναπαράσταση του πεδίου Index για μία διεύθυνση), ανάλογα με τις παραμέτρους που έδωσε ο χρήστης στο αρχείο *Libraries_and_defines.h*. Οι τιμές των defines, αντιγράφονται σε άλλες μεταβλητές έτσι ώστε να μπορούν να χρησιμοποιηθούν σε μετέπειτα στάδιο σε μαθηματικές πράξεις όπως είναι οι λογάριθμοι, στην μορφή την οποία όμως τις θέλουμε (τον τύπο μίας μεταβλητής, π.χ να είναι τύπου Integer). Επίσης σε αυτό το σημείο, πραγματοποιούνται οι περισσότεροι έλεγχοι πάνω στις παραμέτρους που όρισε ο χρήστης. Μερικά παραδείγματα ελέγχων είναι:
   i) Αν τα μεγέθη των μνημών και της λέξης είναι θετικοί αριθμοί και στη δύναμη του 2. (Ο 2ος έλεγχος γίνεται με την βοήθεια της λογικής πράξης AND, μεταξύ της τιμής μίας μεταβλητής και της ίδιας τιμής αλλά μειωμένης κατά 1, βλέποντας κατά πόσο το αποτέλεσμα της πράξης είναι 0 ή όχι)
   ii) Αν ο χρήστης επέλεξε μόνο μία απο τις τρεις διαφορετικές αρχιτεκτονικές. (Direct Map, N-Way Set Associative ή Fully Associative)
   iii) Αν η κρυφή μνήμη είναι μικρότερη από την κύρια μνήμη.

   Αν όλοι οι παράμετροι περάσουν επιτυχώς όλους τους ελέγχους, τότε η σύναρτηση *structures* θα επιστρέψει στο κυρίως πρόγραμμα, τον αριθμό των ψηφίων που χρειάζονται για το κάθε πεδίο, ανάλογα με το τι αρχιτεκτονική ακολουθεί η υλοποίησή μας. Αν όχι, τότε θα εμφανιστεί το κατάλληλο μήνυμα λάθους στην οθόνη, το οποίο θα ενημερώνει τον χρήστη για το πρόβλημα και η εκτέλεση του προγράμματος θα τερματιστεί.
4. Η συνέχεια του προγράμματος βρίσκεται και πάλι στην *main()* συνάρτηση. Στο σημείο εκείνο, εκτελείται ένα for loop το οποίο με τη κατάλληλη χρήση των συναρτήσεων srand() και rand(), δημιουργεί και αποθηκεύει στο αρχείο *DataFile*, ένα σύνολο εντολών και διευθύνσεων (σε δεκαδική μορφή). Το σύνολο των εντολών και διευθύνσεων, καθορίζεται από τις τιμές των παραμέτρων που δηλώνονται στο header file *Libraries_and_Defines*. Γενικά, το πρόγραμμά μας δημιουργεί μια τυχαία εντολή και μια τυχαία διεύθυνση, στη συνέχεια μεταφράζει την δεκαδική αυτή διεύθυνση σε δυαδική και στο τέλος, γράφει όλες αυτές τις πληροφορίες που παράχθηκαν στο αρχείο δεδομένων. Σε αυτό το σημείο του κώδικα είναι υλοποιημένος και ο έλεγχος για το πόσο συχνά μπορεί να γίνεται flush η κρυφή μνήμη, το οποίο αποτελεί μία ανάγκη για την σωστή λειτουργία του προγράμματος.

5. Κάθε φορά που το πρόγραμμα δημιουργεί μια τυχαία διεύθυνση, καλεί και την συνάρτηση *DecimalToBinary* η οποία βρίσκεται στο αρχείο *Converter.h* . Σε αυτό το αρχείο υπάρχει ένας απλός κώδικας, ο οποίος υλοποιεί την μετάφραση της δεκαδικής διεύθυνσης σε δυαδική και αποθηκεύει το αποτέλεσμα bit by bit (0 ή 1 κάθε φορά) σε έναν πίνακα. Αυτόν το πίνακα τον χρησιμοποιούμε στην συνέχεια για να τυπώσουμε την δυαδική διεύθυνση στο αρχείο δεδομένων. Για να γίνει η μετατροπή αυτή, γίνεται η πράξη της διαίρεσης του δεκαδικού αριθμού με τον αριθμό 2, μέχρι να φτάσει στην τελική του μορφή (1) καθώς υπολογίζεται κάθε φορά το υπόλοιπο.
6. Τέλος, το κυρίως πρόγραμμα καλεί την συνάρτηση *parser* η οποία βρίσκεται στο αρχείο *parser.h,* ώστε να διαβάσει το αρχειο δεδομένων και να εκτυπώσει τα ανάλογα μηνύματα στο Output file.
7. Στο αρχείο *Parser.h,* χρησιμοποιούμε δύο διαφορετικούς δείκτες σε αρχεία, έναν στο αρχείο δεδομένων για ανάγνωση και έναν στο αρχείο εξόδου για εγγραφή. Ο parser μας εκτελεί ένα for loop ίδιου μεγέθους με αυτό του κυρίους προγράμματος. Σε κάθε κύκλο του, διαβάζεται ένας χαρακτήρας από το αρχείο δεδομένων. Αρχικά, διαβάζεται ο πρώτος χαρακτήρας που βρίσκεται στην συγκεκριμένη γραμμή, και ανάλογα από το ποιος χαρακτήρας(R: Read, M:Modify, W:Write, F:Flush) είναι, εκτελείται ένα από τα τέσσερα cases. Αν το πρώτο γράμμα ήταν F τότε θα τυπώσουμε στο output file την λέξη FLUSH. Ακολουθεί ένα while το οποίο θα μας μεταφέρει στην αμέσως επόμενη γραμμή για να την διαβάσουμε. Αν όμως το πρώτο γράμμα ήταν W, R ή M τότε εκτυπώνουμε το ανάλογο μήνυμα στο output file (Write, Read, Modify) και χρησιμοποιώντας ένα loop, τυπώνουμε δίπλα την δυαδική διεύθυνση που του αντιστοιχεί, ψηφίο προς ψηφίο. Κάθε φορά που διαβάζουμε ένα ψηφίο της δυαδικής διέυθυνσης από το αρχείο δεδομένων και το μεταφέρουμε στο output file, το αποθηκεύουμε και σε ένα πίνακα. Στη συνέχεια, χρησιμοποιώντας τρία διαφορετικά loops, τυπώνουμε τα ψηφία των πεδίων tag, index και block offset. Αν η αρχιτεκτονική ήταν N-Way set associative τότε θα επιλεγεί τυχαία ένα από τα N ways και θα τυπώνεται παράλληλα και αυτό στο output file, δίπλα από τα πεδία που υπολογίστηκαν. Αυτή η διαδικασία γίνεται για κάθε γραμμή του αρχείου δεδομένων και όταν διαβαστούν επιτυχώς όλες οι γραμμές τους, κλείνουμε τα δύο αρχεία και επιστρέφουμε την εκτέλεση του προγράμματος στο source.c όπου και τελειώνει η εκτέλεση του προγράμματος μας με βάση τις λειτουργίες που ζητήθηκαν για το $1^{ο}$ milestone.

<u>Προβλήματα και Δυσκολίες που Αντιμετωπίσαμε:</u>

- Μία από τις δυσκολίες που αντιμετωπίσαμε ήταν κατά την αναγκαία δημιουργία ενός πίνακα στο header file "*Parser.h*", αφού λόγω του ότι η γλώσσα προγραμματισμού που χρησιμοποιήσαμε ήταν η C, δεν ήταν δυνατό να γίνει ο ορισμός του μεγέθους του πίνακα αυτού, με βάση το συγκεκριμένο αριθμό bit που χρειάζεται μία διεύθυνση. Έτσι, θέσαμε το πρόγραμμα μας να δημιουργεί τον πίνακα αυτό και να του δίνει για μέγεθος μία συγκεκριμένη ακέραια τιμή. Η τιμή αυτή είναι η 128. Αυτό σημαίνει πως αν υπολογισθεί πως οι διευθύνσεις χρειάζονται περισσότερα από 128 bits, θα παρουσιασθεί Bug στον κώδικα.
- Άλλη δυσκολία, ήταν να αποφασισθεί το ποιοι έλεγχοι χρειάζονταν να γίνουν στο πρόγραμμα ώστε να καλύπτουν κάθε πιθανή ακραία τιμή η οποία θα δοθεί στις παραμέτρους. Αυτό ήταν αρκετά χρονοβόρο και κάποιοι από τους ελέγχους απαιτούσαν πολύπλοκη σκέψη για να υλοποιηθούν και να είναι παράλληλα μικροί και εύκολα διαχειρίσιμοι.
- Ένα πρόβλημα, το οποίο αντιμετωπίζαμε συνεχώς και δεν καταφέραμε να βρούμε ποιο ήταν το λάθος που το προκαλούσε, είναι πως κάποιες φορές όταν αποστέλλουμε τον κώδικα μας ο ένας του άλλου, εμφανίζονται ξαφνικά λάθη τα οποία δεν υπήρχαν στην πραγματικότητα. Δηλαδή, ενώ ο κώδικας μας έτρεχε κανονικά στο Visual Studio του υπολογιστή μας, όταν τον στέλνουμε μέσω διάφορων μέσων (π.χ Dropbox), στον συνεργάτη μας, βγάζει λάθη στον κώδικα τα οποία δεν υπάρχουν στον δικό μας κώδικα. **Αυτο συμβαίνει κα όταν ανεβάζουμε (push-pull) τον κώδικα μας μέσω του sourcetree στο github.**

<u>Διαχωρισμός διεργασιών:</u>

- Για την υλοποίηση του προγράμματος, συνεργαστήκαμε σχεδόν σε όλα τα μέρη του. Παρ' όλα αυτά, επειδή δεν ήταν δυνατές οι καθημερινές και πολύωρες συναντήσεις μας, αναγκαστήκαμε να χωρίσουμε και να υλοποιήσουμε ξεχωριστά κάποια κομμάτια του κώδικα ο καθένας μας.
- Συγκεκριμένα , ο Μιχάλης Προδρόμου δούλεψε περισσότερο στο header file "*Structures.h*" και η Μαριαλένα Σιαμμά στο header file "*Parser.h*".
- Όμως, ανταλλάζαμε συνεχώς τους κώδικες μας ώστε όχι μόνο να επιλύσουμε και να συζητήσουμε τυχόν προβλήματα και δυσκολίες που αντιμετωπίσαμε, αλλά και για να κάνουμε ελέγχους για το κατά πόσο το κομμάτι του κώδικα που γράψαμε δούλευε όντως σωστά και δεν επηρέαζε αρνητικά την λειτουργία του υπόλοιπου προγράμματος.
- Όπως φάνηκε, αυτή η μέθοδος εξέλιξης του προγράμματος, αποδείχθηκε πολύ καλή, αφού οι διαδικασίες προχωρούσαν πολύ γρήγορα και δεν βαρυφόρτωνε κάποιο από τα δύο άτομα της ομάδας, αφού όταν δεν ήταν δυνατή μία συνάντηση για κάποιους λόγους, μπορούσε το άλλο μέλος να συνεχίσει την υλοποίηση μόνο του και να συζητήσει μετά με το άλλο μέλος της ομάδας για τις δοκιμές και αλλαγές που έκανε στο πρόγραμμα.