

Τεχνολογικό Πανεπιστήμιο Κύπρου
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ και Πληροφορικής
Φθινοπωρινό Εξάμηνο 2013 - 2014
ΜΗΥΠ 323: Αρχιτεκτονική Η/Υ
Μιχάλης Προδρόμου – Μαριαλένα Σιαμμά
Τελική Εργασία – Αναφορά 2^{ου} 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 παραμέτρων. Βάση αυτών των παραμέτρων, θα γίνονται οι υπολογισμοί για τα πεδία των tag, index και block offset σε μετέπειτα στάδιο.
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` 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 `ParserParameters()`, η οποία βρίσκεται στο αρχείο `ParserParameters.h`.
3. **ParserParameter.h:** Σε αυτό το αρχείο πραγματοποιείται το διάβασμα των παραμέτρων που έδωσε ο χρήστης στο αρχείο παραμέτρων. Αρχικά αποθηκεύονται σε μεταβλητές αυτού του αρχείου και όταν τελειώσει το διαβασμα όλου του αρχείου παραμέτρων θα επιστραφούν στο κυρίως πρόγραμμα (`main()`).
4. Ακολούθως η ροή του προγράμματος μεταφέρεται στο αρχείο **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 παραμέτρων. Οι τιμές τους, 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 2^{ος} 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 από 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 συνάρτηση `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.
5. 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.
6. 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 τον 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.

7. Μετά, το κυρίως πρόγραμμα καλεί την συνάρτηση *parser* η οποία βρίσκεται στο αρχείο *parser.h*, ώστε να διαβάσει το αρχείο δεδομένων και να εκτυπώσει τα ανάλογα μηνύματα στο Output file. Επίσης εδώ υλοποιούνται και όλες οι λίστες από structs ανάλογα με την αρχιτεκτονική που επέλεξε ο χρήστης.
8. 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 κυρίως 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 και θα εκκενώσουμε όλα τα δεδομένα της λίστας και θα θέσουμε ξανά όλα τα valid με 0. 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 διεύθυνσης 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.

Γενικά σε αυτό το αρχείο, υλοποιούνται οι δομές (structs) και οι λίστες (dynamic) για κάθε αρχιτεκτονική. Για παράδειγμα στην Direct Mapped Αρχιτεκτονική, δημιουργούνται τόσοι κόμβοι όσος είναι ο αριθμός των index μας, δηλαδή των Cache entries. Η N-Way Set Associative, λειτουργεί παρόμοια με την 1^η αρχιτεκτονική, δηλαδή κινείται με βάση το Index, αλλά για κάθε index δημιουργούνται τόσοι κόμβοι όσα είναι τα Ways που δόθηκαν στο Parameters file. Αντίθετα στην Fully Associative, δημιουργούνται κόμβοι ανάλογα με το Tag των αναφορών, αφού δεν υπάρχει Index στην αρχιτεκτονική αυτή.

Ακολουθώς γίνονται όλοι οι σχετικοί έλεγχοι που χρειάζονται για να υπολογιστεί αν η αναφορά στην κρυφή μνήμη ήταν επιτυχή, δηλαδή αν υπήρχε το ίδιο (Index,) Tag και η ίδια διεύθυνση στη σωστή θέση του Block Offset του κόμβου ή αποτυχημένη, δηλαδή αν κάτι από τα πιο πάνω διέφερε με αποτέλεσμα ανάλογα από το αν ήταν εντολή Read/Modify ή Write γράφουμε τη σωστή αναφορά διεύθυνσης στην Cache. Μετά γίνεται ο υπολογισμός των κύκλων που χρειάστηκαν, ανάλογα με την πολιτική εγγραφής που ισχύει (FIFO, LRU, Random) και τα δεδομένα που πήραμε από το αρχείο *Parameters_File*. Στη συνέχεια τυπώνουμε στο output file το αν ήταν hit ή miss, τον τύπο του miss και τους κύκλους που χρειάστηκε η συγκεκριμένη αναφορά.

Αυτή η διαδικασία γίνεται για κάθε γραμμή του αρχείου δεδομένων και όταν διαβαστούν επιτυχώς όλες οι γραμμές τους, υπολογίζονται τα στατιστικά δεδομένα που πρέπει να επιστρέφονται και να εμφανίζονται στην οθόνη. Τέλος 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 2^ο milestone.

Προβλήματα και Δυσκολίες που Αντιμετωπίσαμε:

- Πολύ περιορισμένος χρόνος για την υλοποίηση και των τριών Αρχιτεκτονικών
- Έπρεπε να υπάρχει πολύ καλή κατανόηση της θεωρίας ακόμη και σε πολύ μεγάλες λεπτομέρειές της
- Λόγω του ότι έγινε προσέγγιση της Cache με τέτοιο τρόπο ώστε να μην υπάρξει υπερβολικά μεγάλος αριθμός λειτουργιών τις οποίες έπρεπε να υλοποιήσουμε, κάποια δεδομένα που είχαμε δεν ισχυαν όπως είναι στην πραγματικότητα, οπότε υπήρχε μία σύγχυση για το τι έπρεπε να γίνει σε κάποιες περιπτώσεις (π.χ1 Write Back (= γράφει και στη Cache και στην RAM & No Write Allocate (=γράφει μόνο στην RAM, π.χ2 Το ότι στις Read και Modify εντολές όταν υπάρξει miss, δεν φέρνουν δεδομένα στην Cache από την δευτεροβάθμια μνήμη –RAM-)
- Ισχύουν επίσης κάποιες από τις δυσκολίες που υπήρξαν και στο πρώτο Milestone

Διαχωρισμός διεργασιών:

- Όπως και στο 1ο milestone, αδυνατούσαμε να κάνουμε ικανοποιητικό αριθμό συναντήσεων λόγω διαφορετικών προγραμμάτων και έλλειψης χρόνου από τα μέλη της ομάδας, αναγκαστικά μοιράσαμε τα μέρη του προγράμματος με τα οποία έπρεπε να ασχοληθεί ο καθένας μας. Και τα δύο μέλη τις ομάδας ασχολήθηκαν και με τις δύο βασικές αρχιτεκτονικές οι οποίες υλοποιήθηκαν. Συγκεκριμένα ο Μιχάλης Προδρόμου δούλεψε περισσότερο στις λειτουργίες Write της Direct Mapped και των Read και Modify της Fully Associative ενώ η Μαριαλένα Σιαμμά δούλεψε περισσότερο στις εντολές Read και Modify της Direct Mapped και της Write της Fully Associative. Η 3^η αρχιτεκτονική υλοποιήθηκε και από τους δύο μας, αλλά λόγω έλλειψης χρόνου δεν ολοκληρώθηκε. Παρ' όλα αυτά υπήρχε συνεχής επικοινωνία μεταξύ των μελών τη ομάδας.