ΤΗΜΜΥ ΑΠΘ Μικροεπεξεργαστές και Περιφερειακά (1ηΕργασία) 2022

Ομάδα 10: Καραμητόπουλος Παναγιώτης ΑΕΜ 9743 karamitopp@ece.auth.gr

Σαριδάκη Μαρία-Ραφαηλία ΑΕΜ 9633 saridakm@ece.auth.gr

Σύντομη Περιγραφή: Σκοπός της 1ης εργασίας είναι η υλοποίηση μιας ρουτίνας, σε assembly ARM, η οποία θα υπολογίζει το hash του αλφαριθμητικού, θα αποθηκεύει την τιμή του σε μια θέση μνήμης και θα την επιστρέφει, καθώς και η υλοποίηση της βασικής ρουτίνας (main) σε γλώσσα C, η οποία θα παρέχει το αλφαριθμητικό που θα ελεγχθεί, θα καλεί την συνάρτηση που υλοποιήθηκε σε ARM assembly, και θα εκτυπώνει με την χρήση της printf σε επίπεδο προσομοίωσης, το αποτέλεσμα.

Υλοποίηση Ρουτίνας (ARM Assembly): Η __asm int hash_calculate(char *string, uint8_t *array, int *hash) είναι η ρουτίνα υλοποιημένη σε ARM Assembly, η οποία υπολογίζει το hash του αλφαριθμητικού. Έχει ως ορίσματα το char *string, το οποίο είναι ένας δείκτης που δείχνει στον πρώτο χαρακτήρα του αλφαριθμητικού, για το οποίο θα υπολογισθεί το hash, το uint8_t *array ο οποίος είναι ένας δείκτης στον πίνακα array, καθώς και το int *hash ο οποίος είναι ένας δείκτης που δείχνει στη θέση μνήμης όπου η ρουτίνα θα αποθηκεύσει την τιμή του hash. Ο μονοδιάστατος πίνακας array περιέχει τον αριθμό, ο οποίος θα προστίθεται στο hash για το αντίστοιχο κεφαλαίο λατινικό γράμμα (A →array[0] = 10, B →array[1] = 42,…, Z→array[25] = 25).

Ο καταχωρητής r0 είναι ο δείκτης προς το αλφαριθμητικό, ο καταχωρητής r1 είναι ο δείκτης προς τον πίνακα array, ενώ ο r2 είναι ο δείκτης στη διεύθυνση μνήμης που θα αποθηκευτεί το hash.

Για την υλοποίηση της ρουτίνας χρησιμοποιήθηκαν και οι καταχωρητές r4, r5 (preserved register) άρα πρέπει να αποθηκεύσουμε την αρχική τους τιμή στο stack. Στον καταχωρητή r4, αποθηκεύονται οι διάφοροι υπολογισμοί του hash, στον r5 αποθηκεύονται κάποιοι υπολογισμοί και στον r3 αποθηκεύτηκε η ASCII τιμή του χαρακτήρα του string που μας ενδιαφέρει σε κάθε επανάληψη.

Αρχικοποιούμε την προσωρινή τιμή του hash στο 0 (δηλαδή κάνουμε mov στον r4 το 0). Κατόπιν υλοποιήθηκε ένα κομμάτι κώδικα με label -> start. Εκεί φορτώνεται στον καταχωρητή r3 το περιεχόμενο της μνήμης στην διεύθυνση r0.

Έπειτα γίνονται οι εξής συγκρίσεις:

- ✓ Συγκρίνεται ο χαρακτήρας που έχει αποθηκευτεί στον r3 με τον χαρακτήρα ‘0’(ή 48) στον πίνακα ASCII. Αν ο χαρακτήρας του r3 είναι μικρότερος ή ίσος από τον χαρακτήρα ‘0’, τότε μεταβαίνει στο label next
- ✓ Συγκρίνεται ο χαρακτήρας που έχει αποθηκευτεί στον r3 με τον χαρακτήρα ‘Z’ (ή 90) στον πίνακα ASCII. Αν ο χαρακτήρας του r3 είναι μεγαλύτερος από τον χαρακτήρα ‘Z’, τότε μεταβαίνει στο label next
- ✓ Συγκρίνεται ο χαρακτήρας που έχει αποθηκευτεί στον r3 με τον χαρακτήρα ‘9’ (ή 57) στον πίνακα ASCII. Με τις εντολές ADDLS & SUBLS, αν ο χαρακτήρας αυτός είναι μικρότερος ή ίσος από τον χαρακτήρα ‘9’ (άρα ο χαρακτήρας είναι αριθμητικό ψηφίο), τότε προστίθεται αρχικά στο hash ο αριθμός 48 και αφαιρείται από το hash ο αριθμός που αντιστοιχεί στον κωδικοποίηση ASCII του εκάστοτε αριθμητικού ψηφίου. Έτσι για τον χαρακτήρα ‘1’(ή 49 στον ASCII) ο τελικός αριθμός που θα αφαιρεθεί από το hash, θα είναι το 49 – 48 = 1, για τον χαρακτήρα ‘2’ (ή 50 στον ASCII) ο τελικός αριθμός που θα αφαιρεθεί από το hash, θα είναι το 50 – 48 = 2, κ.ο.κ. Τέλος αν ισχύει η παραπάνω συνθήκη μεταβαίνει στο label next.

- ✓ Συγκρίνεται ο χαρακτήρας που έχει αποθηκευτεί στον r3 με τον χαρακτήρα '@' (ή 64) στον πίνακα ASCII (ο χαρακτήρας @ είναι ο χαρακτήρας πριν τον χαρακτήρα 'A'). Αν ο χαρακτήρας του r3 είναι μικρότερος ή ίσος από τον χαρακτήρα '@', τότε μεταβαίνει στο label next
- ✓ Αλλιώς ο χαρακτήρας θα είναι κεφαλαίο λατινικό γράμμα (A-Z), αφαιρούμε το 65 (το 'A' στον ASCII, έτσι ώστε να έχουμε την αντιστοίχηση: (A →array[0] = 10, B →array[1] = 42,…, Z→array[25] = 25) , από τον καταχωρήτη r3, ούτως ώστε το αποτέλεσμα που αποθηκεύεται στον r5 να είναι η αντίστοιχη θέση του hash του χαρακτήρα στον πίνακα array. Φορτώνουμε στον r5 το περιεχόμενο της διεύθυνσης μνήμης r1 + r5, και το προσθέτουμε στο hash. Έπειτα μεταβαίνει στο label next

Τέλος, στο label next, αυξάνεται η τιμή του δείκτη στο string κατά 1. Έπειτα γίνεται η σύγκριση του χαρακτήρα που διαβάστηκε με το 0, ('NUL' στον ASCII, ο οποίος χρησιμοποιείται για να δείξει ότι το string τελείωσε). Αν ο χαρακτήρας δεν είναι 0, γίνεται branch στο label start, ενώ αν είναι 0, το αλφαριθμητικό μας έχει τελειώσει, γίνεται mov το hash (r4) στον r0 προκειμένου η ρουτίνα να επιστρέψει το hash. Στη συνέχεια το hash (r4) γράφεται στην θέση μνήμης που το ορίζει το 3ο όρισμα της ρουτίνας, ο pointer *hash. Έπειτα ανακτώνται ο r4 και ο r5, ώστε να αποκτήσουν την αρχική τους τιμή. Τέλος γίνεται η επιστροφή από την ρουτίνα.

Υλοποίηση Main (σε C): Στην main, δηλώνεται το αλφαριθμητικό, και αρχικοποιείται η μεταβλητή όπου θα αποθηκευτεί το hash. Καλείται η ρουτίνα που υπολογίζει το hash, και τυπώνεται με τη χρήση της printf το δοθέν αλφαριθμητικό και το hash που προκύπτει μέσω της ρουτίνας hash_calculate.

Προβλήματα που αντιμετωπίστηκαν: Το keil εμφάνιζε errors στην δήλωση της ρουτίνας σε ARM assembly **"error: expected '(' after 'asm' ' και ' error: expected ';' after top-level asm block"** , για την επίλυση του προβλήματος αλλάξαμε τον compiler, στην Version 5, αντί της 6. Με την έκδοση 5 συνεχίζει το Dynamic Syntax Checking να εμφανίζει τα ίδια errors αλλά σύμφωνα με το support του Keil η Version 6 του compiler χρησιμοποιεί Dynamic Syntax Checking, χρησιμοποιώντας την Version 5 έγινε build αγνοώντας αυτά τα errors.

Το δεύτερο πρόβλημα που αντιμετωπίστηκε ήταν ότι τα label έπρεπε να είναι τέρμα αριστερά στον κώδικα μας, χωρίς κάποιο κενό, μετά από πολλές αποτυχημένες προσπάθειες το πρόβλημα εντοπίσθηκε-επιλύθηκε. Αν το label ήταν κατά ένα tab πιο μέσα εμφάνιζε errors και δεν μπορούσε να γίνει build.

Το τρίτο πρόβλημα εμφανίστηκε στο τελευταίο κομμάτι του testing, όπου χρησιμοποιήθηκε ένα string array. Σε αυτή την περίπτωση, για να γίνει build χωρίς errors στο keil, έπρεπε να αλλαχθεί μια ακόμα ρύθμιση (`Magic Wand --> C/C++ --> Check 'C99 Mode' on the right to confirm`).

Testing: Για τον έλεγχο της ορθότητας της εργασίας, δοκιμάστηκαν αρχικά διάφορα αλφαριθμητικά, για τα οποία με τον debugger ελέγχθηκαν step by step οι τιμές των καταχωρητών, ορισμάτων, μεταβλητών και των θέσεων μνήμης κάθε μεταβλητής/ορίσματος. Στη συνέχεια, φτιάχτηκε ένας πίνακας από strings που να καλύπτει διάφορες περιπτώσεις, ως ένας δεύτερος, πιο γρήγορος, τρόπος να δοκιμαστεί ότι ο κώδικας δουλεύει σωστά. Ο πίνακας a[8][10] = {"5", "A", "A5", "h", "h4A", "!", "25ATx0kB", "Test!"} περιέχει strings που έχουν μόνο χαρακτήρες αριθμών, μόνο χαρακτήρες πεζών και κεφαλαίων γραμμάτων, μόνο χαρακτήρες συμβόλων και συνδυασμούς όλων των παραπάνω. Τέλος δημιουργήθηκε και ένα αλφαριθμητικό που περιλαμβάνει σχεδόν όλους του χαρακτήρες του ASCII. Για όλες τις δοκιμές, τα αποτελέσματα ήταν τα επιθυμητά.