Μικροεπεξεργαστές και Περιφερειακά

Εργασία 1

1) Εισαγωγικά

Σε αυτήν την εργασία, ήταν η πρώτη μου επαφή με την ARM assembly, καθώς και με το εργαλείο Keil. Αναφορικά με την εκφώνηση, μας ζητήθηκε να υλοποιήσουμε έναν έλεγχο-αναγνώριση σε κάποιο τυχαίο string που θα αρχικοποιούσαμε, ούτως ώστε αν έχουμε σε αυτό το string κεφαλαίους λατινικούς χαρακτήρες να προσθέτουμε κάποιον αριθμό σε κάποιο sum βάσει του πίνακα που δόθηκε, ή αν έχουμε κάποιον αριθμό (για χαρακτήρα) να τον αφαιρούμε από το sum. Στο sum υπολογίζεται το hash. Ένας τρόπος θα ήταν να ελέγχουμε όλα τα δυνατά σενάρια, δηλαδή το αν ο κάθε χαρακτήρας του string που δοκιμάζουμε είναι ίσος με κάποιον από τους 26 κεφαλαίους λατινικούς χαρακτήρες ή ίσος με τους χαρακτήρες από το 1 έως το 9. Αυτό θα μας οδηγούσε στο να κάνουμε σύνολο 35 ελέγχους για κάθε χαρακτήρα του string, σε ένα worst-case scenario. Το δοκίμασα και αυτό και λειτουργεί κανονικά.

2) Σχετικά με την υλοποίηση

Αντί να κάνω το παραπάνω, επέλεξα να προσδιορίσω τον κάθε χαρακτήρα διαφορετικά. Παρατήρησα ότι κάθε χαρακτήρας που αρχικοποιούσα έμπαινε στους registers ως μια τιμή σε hex. Συνεπώς, παίρνοντας την διαφορά (των τιμών σε hex) των χαρακτήρων που δοκιμάζουμε, έναν προς έναν, με κάποιους χαρακτήρες ως σημεία αναφοράς, μπορούμε να προσδιορίσουμε πιο γρήγορα και αποτελεσματικά το ποιοι χαρακτήρες είναι. Πιο συγκεκριμένα, επειδή ο χαρακτήρας A σε hex είναι το 0x41 και ο Z το 0x5A (δηλαδή από το 65 έως το 90 σε dec) και έχοντας ένα αρχικοποιημένο string για δοκιμή, έλεγξα τα γράμματα ένα προς ένα αφαιρώντας το A από τον χαρακτήρα του string μου. Έτσι θα προκύψει ένας νέος αριθμός της διαφοράς αυτής. Έπειτα αυτήν τη διαφορά θα τη συγκρίνω με το 0 και θα έχω 2 περιπτώσεις :

i) Αν η διαφορά αυτή είναι μικρότερη του 0, αυτό σημαίνει ότι ο χαρακτήρας που αφαιρέσαμε από το A βρίσκεται κάτω από το A στον πίνακα ASCII, σε hex. Στην καλύτερη (για εμάς) περίπτωση, είναι κάποιος αριθμός (το 1 είναι το 0x31 και το 9, το 0x39). Συνεπώς θα ελέγξω αν είναι αριθμός, καθώς μόνο αν είναι τέτοιος με ενδιαφέρει. Ο έλεγχος αυτός γίνεται με παρόμοια λογική με την παραπάνω. Πιο συγκεκριμένα, παίρνω εκ νέου τη διαφορά του ίδιου χαρακτήρα που δοκιμάζω, αυτήν τη φορά με τον χαρακτήρα 0 (0x30). Συνεπώς πλέον το πεδίο που μας ενδιαφέρει για να κάνουμε κάτι στο hash, θα είναι αυτή η διαφορά να έχει κάποια τιμή από 1 έως και 9. Η διαφορά, θα έχει ακριβώς την τιμή του αριθμού (1 έως 9) και όχι του χαρακτήρα του αριθμού (0x31-0x39). Συνεπώς θα τη συγκρίνω με τους αριθμούς και όχι με τιμές των χαρακτήρων των αριθμών. Έτσι συγκρίνω πρώτα να είναι μεγαλύτερο του 9 και αν είναι τον πετάω και πάω στον επόμενο χαρακτήρα του string (έχω ήδη δει ότι δεν είναι κεφαλαίο γράμμα παραπάνω), ή αν είναι μικρότερο του 1, που και πάλι πάω στο next. Αν περάσει αυτούς τους ελέγχους, η διαφορά θα είναι αναγκαστικά αριθμός από 1 έως 9, έτοιμος να αφαιρεθεί κατευθείαν από το hash χωρίς καμία περαιτέρω τροποποίηση.

ii) Αν η διαφορά με τον χαρακτήρα A είναι μεγαλύτερη του 0, μπορεί το στοιχείο του string που ελέγχω να είναι κεφαλαίος χαρακτήρας, αρκεί να ελέγξουμε ότι η τιμή της διαφοράς αυτής δεν είναι μεγαλύτερη από τον αριθμό 25. Αυτό το κάνω, καθώς όπως είπα και παραπάνω, υπάρχουν 26 κεφαλαίοι λατινικοί χαρακτήρες που βρίσκονται διαδοχικά στον πίνακα ASCII. Συνεπώς η αφαίρεση με τον χαρακτήρα A, του στοιχείου του string που ελέγχουμε, αν έχει τιμή από 0 έως και 25 θα είναι αναγκαστικά κάποιος κεφαλαίος χαρακτήρας (0 για A, 25 για Z σύνολο 26 στοιχεία). Αυτήν την τιμή, την χρησιμοποίησα σαν δείκτη σε ένα πίνακα b[26], ο οποίος έχει integer στοιχεία και έχει διαδοχικά τις τιμές που πρέπει να προσθέσουμε στο hash, για κάθε γράμμα. Συνεπώς και αυτή η διαφορά, χρησιμοποιείται στην περίπτωση που είναι από 0 έως και 25 σαν δείκτης του πίνακα b, αφού πρώτα πολλαπλασιαστεί με το 4, καθώς ο b έχει integers για στοιχεία που είναι 4 bytes, οπότε και για να γίνει προσπέλαση σε αυτά θα πρέπει να δείχνουμε στην αρχή των 4 αυτών bytes. Για παράδειγμα, αν είναι ίση με 0, θα πάρω το b[0] στοιχείο που είναι το 18 και θα το προσθέσω στο hash.

Μετά από αυτήν τη διαδικασία, προχωράμε στο επόμενο στοιχείο μέχρι να τελειώσει το string που ελέγχουμε. Όταν τελειώσει το string θα έχει υπολογιστεί και το ζητούμενο.

3) Σχετικά με τον κώδικα

Όσον αφορά τη main, χρησιμοποίησα 2 πίνακες, έναν const char a[] για να αρχικοποιώ το string που θα ελέγξω και έναν int b[26], που περιέχει με τη σειρά όλες τις τιμές του πίνακα που μας δόθηκε στην εκφώνηση για να τις προσθέτουμε στο hash. Ακόμη αρχικοποιώ μια μεταβλητή int lengthA, που παίρνει το μήκος του πίνακα a[] με τη συνάρτηση strlen() του header file string.h (που χρειάζεται και type cast σε int). Τέλος, αρχικοποιώ το result, καλώ την συνάρτηση/ρουτίνα hash και τυπώνω το αποτέλεσμα.

Αναφορικά με την hash (const char *a, int *b, int c, int d), είναι η ρουτίνα που μας ζητήθηκε να υλοποιήσουμε και αναφέρομαι στη λογική και παραπάνω. Εδώ περνάω 4 ορίσματα, και πιο συγκεκριμένα, τον πίνακα a[], τον πίνακα b[], το lengthA και το result. Θα μπορούσαμε να μην περάσουμε το 4^{ο} όρισμα και απλά να αρχικοποιούμε μέσα στην ρουτίνα της assembly κάποιον register στον οποίο θα υπολογιστεί το hash, αλλά περνώντας το result, μπορούμε μετά να το χρησιμοποιήσουμε και στη main στην οποία και γυρνάει. Παρατήρησα ότι όταν καλείται η συνάρτηση hash, τα ορίσματα μπαίνουν με τη σειρά στους r0-r3 (a[] → r0, b[] → r1, lengthA → r2, result → r3). Αν είχαμε παραπάνω ορίσματα, θα μπαίνανε στον stack pointer, καθώς ο compiler μπορεί να χρησιμοποιήσει από τους general purpose, τους r0-r3. Παρ' όλα αυτά, το result, όταν γυρίσουμε στη main, θα είναι στον register που αρχικοποιήθηκε στην main, δηλαδή στον r4. Ακόμη όσον αφορά τους registers, αρχικοποίησα μέσα στην hash και τον r8 ως τον αριθμό 4, καθώς ο πίνακας b[] έχει integers που είναι 4 bytes το καθένα, οπότε θα πρέπει να πολλαπλασιάζουμε τον δείκτη του b[] με το 4 και η εντολή MUL δεν παίρνει απευθείας κάποιον αριθμό, οπότε θα βάζω τον r8. Επίσης, παρ' όλο που πέρασε στο r3 το result, ξέρουμε ότι θα βγει στο r4. Οπότε ο r3 χρησιμοποιείται μέσα στη συνάρτηση σαν ένα increment i το οποίο είναι ήδη αρχικοποιημένο, για να γίνεται προσπέλαση στον πίνακα a[], ο οποίος έχει στοιχεία char, που είναι 1 byte, οπότε και δεν χρειάζεται να τον πολλαπλασιάσουμε με κάποιον αριθμό. Στον r4 υπολογίζεται το result, στον r5 κάνουμε load byte το στοιχείο a[i], δείχνοντας στον pointer του πίνακα μετατοπισμένο κατά i [r0, r3]. Οι διαφορές που ανέφερα παραπάνω υπολογίζονται στον r6, και χρησιμοποιούνται και σαν increment j του πίνακα b[]. Στο r7 κάνω load byte το στοιχείο b[j] που περιέχει την τιμή που πρέπει να αφαιρέσουμε από το hash. Σημαντική σημείωση είναι, ότι ανάμεσα σε αυτά γίνονται και οι έλεγχοι με CMP, οπότε αν ο χαρακτήρας που εξετάζουμε δεν ανήκε σε αυτά που μας ενδιαφέρουν, απλά θα πάμε να ελέγξουμε τον επόμενο χαρακτήρα. Καλό είναι να υπάρχει και ένα breakpoint στο return 0, ούτως ώστε να φαίνεται και στον r4 το αποτέλεσμα και στο debug printf viewer το print.

Μετά την αρχικοποίηση του r8, ένα branch στην αρχή ελέγχει αν το string είναι κενό ή όχι. Έπειτα ο κώδικας είναι χωρισμένος σε 4 κομμάτια, το check_cap, που ελέγχει για κεφαλαίους λατινικούς χαρακτήρες, το check_num, που καλείται από το check_cap και ελέγχει για τους αριθμούς 1 έως 9, το next που απλά αυξάνει το increment i που κάνει access τον a[] και τέλος το limit που βλέπει αν διαβάσαμε όλο το string a[], ελέγχοντας το increment i < lengthA και επιστρέφει στη main με BX lr, αν γίνουν ίσα.

4) Παρατηρήσεις/Σκέψεις σχετικά με την εργασία

Το Keil σαν πρόγραμμα δεν είχε κάποια ιδιαίτερη δυσκολία. Αν βάζουμε σωστά τα packages για τη σωστή πλακέτα, τις ρυθμίσεις για default compiler 5, το να χρησιμοποιούμε τον simulator για debug και κάποια πακέτα για το project στο runtime environment, έπειτα δεν έχουμε κάποιο ιδιαίτερο θέμα. Σχετικά με την printf ακολούθησα το pdf που υπάρχει στο elearning από κάποιους συναδέλφους, απλά με simulator ως debug. Σχετικά με τον κώδικα, είδα κάποια παραδείγματα που χρησιμοποιούσαν στο τέλος της main while(1) αντί για return 0. Με while(1) γινόντουσαν όλα καλά αλλά δεν τυπώνονταν στο debug printf viewer το αποτέλεσμα. Αναφορικά με τους registers, έχω την εντύπωση ότι αν γράφαμε ολόκληρο το πρόγραμμα σε assembly, θα έπρεπε κάποια MOV και PUSH κλπ κλπ που κάνει αυτόματα το keil να τα κάναμε μόνοι μας (πχ για να περάσουμε ορίσματα στην assembly). Τέλος, με προβλημάτισε το αν είναι καλή πρακτική το να χρησιμοποιούμε κατευθείαν τους r0-r3, καθώς μάλλον αν είχαμε κάποια υπορουτίνα μέσα στην ρουτίνα, που είχε ορίσματα, θα γινόντουσαν overwrite οι registers αυτοί. Συνεπώς ίσως είναι πιο σωστό να μεταφέρουμε τα ορίσματα αυτά σε άλλους registers σε πιο περίπλοκους κώδικες.

Ηλιάδης-Αποστολίδης Δημοσθένης 8811