Αναφορά:

1. **Περιγραφή κώδικα**

Στην πρώτη εργασία καλούμαστε να υλοποιήσουμε μια συνάρτηση σε assembly ARM, η οποία θα υπολογίζει αν το αλφαριθμητικό είναι παλίνδρομο ή όχι, θα επιστρέφει 0 ή 1 στη main (αν δεν είναι ή είναι παλίνδρομο το αλφαριθμητικό αντίστοιχα) και θα αποθηκεύει την ανάλογη τιμή σε μια θέση μνήμης, καθώς και μια main σε C όπου θα ορίζεται το αλφαριθμητικό στατικά.

Η **assembly** συνάρτηση για την αξιολόγηση του string ονομάστηκε **check(),** ο τύπος επιστροφής της είναι int και δέχεται ένα όρισμα πίνακα χαρακτήρων (δηλαδή το string που θέλουμε να αξιολογήσουμε).

Ξεκινώντας, στην check() χρησιμοποιήθηκε η εντολή *PUSH* για να μπουν στο stack τα δεδομένα που είχαν οι καταχωρητές r4 έως r10, με σκοπό να επαναφερθούν κατά την έξοδο απο τη συνάρτηση αυτή.

Ακολουθεί μια λούπα που, όπως φαίνεται και απο το όνομα του *label* της, πρόκειται να μετρήσει το μέγεθος του string που δώσαμε ως όρισμα κατά την κλήση της check(). Τα βήματα που ακολουθούνται εξηγούνται με σχόλια δίπλα απο τις εντολές, αλλά πρακτικά αυτό που γίνεται είναι ότι ξεκινώντας από την αρχή του string (απο τη θέση μνήμης που βλέπει ο r0 καταχωρητής) αποθηκεύουμε ένα ένα τα γράμματά του στον καταχωρητή r2 (τον οποίο σε κάθε επανάληψη της λούπας, συγκρίνουμε με το 0) και όταν φτάσει ο r0 να «κοιτάξει» στη θέση μνήμης που το περιεχόμενό της είναι η τιμή μηδέν, καταλαβαίνουμε ότι φτάσαμε στο τέλος του string. Σε κάθε επανάληψη λοιπόν αυξάνουμε τον r0 κατά ένα (κοιτάμε δηλαδή στο επόμενο γράμμα του string), και όσο το περιεχόμενο που «κοιτάει» ο r0 δεν είναι ίσο με το μηδέν, αυξάνουμε έναν μετρητή (στην προκειμένη αυξάνουμε την τιμή του r1). Κατά την έξοδο απο τη λούπα, οι 2 *SUB* που ακολουθούν, αναλαμβάνουν να επαναφέρουν τον r0 στην πρώτη θέση του string.

Ύστερα ακολουθεί μια νέα λούπα με *label* loop, όπου στο τέλος της θα έχουμε αποφανθεί για το αν το string είναι παλίνδρομο ή όχι. Η διαδικασία που ακολουθείται είναι η εξής:

- Το περιεχόμενο της θέσης μνήμης που έχει ο r0, αποθηκεύεται στον καταχωρητή r6.
- Γίνεται κατάλληλο *offset* στον r0 ώστε να κοιτάξει στη θέση του string που μας ενδιαφέρει στην εκάστοτε επανάληψη του loop, και να αποθηκεύσουμε αυτό το γράμμα στον r7, με την εξής λογική:
  Στην πρώτη επανάληψη, ο r0 «κοιτάει» στην αρχή του string, άρα η τιμή του offset ισούται με το size του string μειωμένο κατά ένα. [A] [N] [N] [A] -> size 4
  r0 (r0+4-1)

Άρα καταφέραμε να αποθηκεύσουμε το πρώτο και το τελευταίο γράμμα σε 2 καταχωρητές και να τα συγκρίνουμε κατα τη διάρκεια της πρώτης επανάληψης.

Στη δέυτερη επανάληψη, ο r0 είχε κινηθεί ένα βήμα δεξιά (κατά την πρώτη επανάληψη), άρα η τιμή του offset χρειάζεται να μειώνεται (μέσα σε κάθε επανάληψη) κατά 2, άρα τώρα έχουμε κάτι τέτοιο: [A] [N] [N] [A]
r0' (r0'+4-1-2)
και έτσι συγκρίνουμε το δεύτερο γράμμα με το προτελευταίο.
Στην Τρίτη προκύπτει κάτι τέτοιο: [A] [N] [N] [A] κτλ..
(r0''+4-3-2) r0''
(Σε κάθε επανάληψη, όταν πάρουμε την πληροφορία που θέλουμε κάνοντας offset τον r0, τον επαναφέρουμε πάλι στη θέση του κάνοντας r0=((r0+offset) – offset)

- Σε κάθε επανάληψη του loop, σε περίπτωση που τα γράμματα που βρίσκονται στον r6 και r7 είναι ίδια, αυξάνουμε έναν μετρητή (συγκεκριμένα τον καταχωρητή r9).
- Οι επαναλήψεις του loop καθορίζονται από τον καταχωρητή r4, ο οποίος ξεκινάει από το 0, και αυξανόμενος κατά ένα σε κάθε επανάληψη, φτάνει την τιμή του size του string (την τιμή που έχουμε στον r5), και τότε καταλαβαίνουμε ότι πρέπει να βγούμε απο τη λούπα.

Κατά την έξοδο απο τη λούπα, αν ο καταχωρητής που μετράει τα matches των r6,r7 ισούται με την τιμή του size, καταλαβαίνουμε ότι το string που δώθηκε ήταν παλίνδρομο.

Συνεχίζοντας, ανάλογα αν ήταν ή όχι παλίνδρομο το string, βάζουμε στον r0 την τιμή 1 ή 0 αντίστοιχα, κάνοντας έτσι το return στη main, που ζητήθηκε, και αποθηκεύουμε και σε μια fixed θέση μνήμης αυτό το αποτέλεσμα. → `MOV32 r3, #536890000 /`
`STRB r0, [r3]//at the`

Τέλος, επαναφέρουμε τις τιμές που είχαμε στους r4-r10 (με την εντολή *POP*) και επιστρέφουμε το πρόγραμμά μας (BX lr)

2. Testing

Για να διαπιστώσω ότι ο κώδικας δουλεύει σωστά, έγιναν δοκιμές με διάφορες λέξεις (παλίνδρομες και μη) και διαπίστωσα ότι κατά την έξοδο της συνάρτησης check, ο r0, που όπως αναφέρθηκε στο τέλος της check παίρνει την τιμή 0 ή 1, ανάλογα αν δεν είναι ή είναι το string παλίνδρομο αντίστοιχα, είχε κάθε φορά την κατάλληλη τιμή.
(Σε string του τύπου hahah, έπαιρνε 1, σε λεξεις hAhah, hello… έπαιρνε μηδέν.)

Επίσης κατά τη διάρκεια του loop, όπου αποθήκευα τα προς σύγκριση γράμματα στους r6,r7 σε κάθε επανάληψη έβλεπα στον debugger τις τιμές αυτών των καταχωρητών, και λάμβαναν όντως τα γράμματα που περίμενα, σε κάθε επανάληψη.

Τέλος για να δω ότι μπήκε η τιμή 1 ή 0 στην fixed θεση μνήμης που επέλεξα (σχετικά random) ελεγξα αν αυτή η θέση μνήμης είχε την τιμή που έπρεπε, μέσω της επιλογής για εμφάνιση του memory window στο keil

3. Προβλήματα που αντιμετώπισα

Το μόνο πρόβλημα που αντιμετώπισα ήταν αυτό που έλυσε το αρχείο Initialization file στο elearning