Μικροεπεξεργαστές και Περιφερειακά
Αναφορά – Εργασία 1

- Όνομα: Στασινός Αλκιβιάδης
- ΑΕΜ: 9214

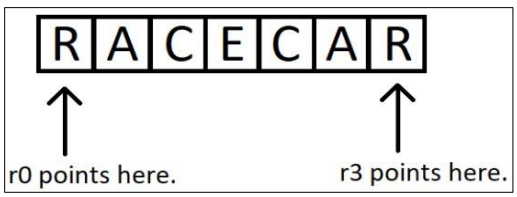
- **Περιγραφή λειτουργίας κώδικα**

Η συγκεκριμένη υλοποίηση περιλαμβάνει δύο συναρτήσεις, τη **main** συνάρτηση γραμμένη στη **C** και τη συνάρτηση **ispalindrome(const char *s)** γραμμένη σε **assembly.** Αρχικά στη **main** ορίζεται ένα αλφαριθμητικό **const char *str = "racecar".** Ύστερα καλείται η συνάρτηση **ispalindrome** δίνοντας της ως argument το αλφαριθμητικό αυτό, ενώ το αποτέλεσμα της (**0 ή 1**) αποθηκεύεται σε μια μεταβλητή τύπου **int** στη **main** και σε μια διεύθυνση μνήμης, συγκεκριμένα στη **0x20000562.**

Όπως φαίνεται και παραπάνω η συνάρτηση **ispalindrome** παίρνει μόνο ένα όρισμα, το αλφαριθμητικό, και η υλοποίηση της μπορεί να χωριστεί σε 3 βασικά **Blocks**. Σημειώνεται επίσης πως για λόγους optimization χρησιμοποιήθηκαν μόνο οι καταχωρητές **R0-R3**, άρα δε χρειάστηκε να σωθεί κάτι στο stack.

1. Πρώτο block με όνομα **STRLEN**.
   Στο block αυτό υπολογίζεται το μήκος του αλφαριθμητικού. Το μήκος θα αποθηκευτεί στον καταχωρητή **R3,** γι' αυτό και πρώτα τον αρχικοποιούμαι με την τιμή **0** και γνωρίζουμε πως στον **R0** υπάρχει η διεύθυνση του πρώτου στοιχείου του αλφαριθμητικού λόγω της κλήσης της από τη **main**. Στη συνέχεια ακολουθεί ένα **loop** με αρχή το label **_COUNT**, όπου πρώτα με την εντολή ***LDRB r1, [r0, r3]*** φορτώνεται στον **R1** το περιεχόμενό της μνήμης που αντιστοιχεί στη διεύθυνση **r0 + r3**. Στο πρώτο **iteration** η τιμή του **R3** είναι **0** και η προηγούμενη εντολή είναι ισοδύναμη με ***LDRB r1, [r0]*** φορτώνοντας στον **R1** τον πρώτο χαρακτήρα. Αν ο χαρακτήρας που φορτώθηκε δεν είναι το **null byte**, τότε ο **R3** θα αυξηθεί κατά ένα και επομένως στο δεύτερο **iteration** η εντολή φόρτωσης θα είναι ισοδύναμη με ***LDRB r1, [r0, #1]*** φορτώνοντας τον επόμενο χαρακτήρα στη σειρά κ.ο.κ. . Ό έλεγχος για το **null** γίνεται αμέσως μετά την ***load*** με τη σύγκριση του **R1** με **0 (CMP r1, #0),** ενώ αμέσως μετά χρησιμοποιούνται οι εντολές ***ADDNE r3, r3, #1*** για **conditional** αύξηση του **R3 μόνο** αν δεν έχει τεθεί το **zero flag** από την ***CMP*** (δηλαδή αν ο **R1 δεν** περιέχει null byte) και φυσικά η ***BNE _COUNT*** για τη συνέχιση του *loop* εάν **δεν** ανιχνεύτηκε ακόμη null byte.

2. Δεύτερο και κύριο block **_START**.
   Στο σημείο αυτό ο **R3** περιέχει το μήκος του αλφαριθμητικού και ο **R0** τη διεύθυνση αρχής του. Εκτελώντας την εντολή ***ADD r3, r3, r0*** ο **R3** θα περιέχει τη διεύθυνση που βρίσκεται το **\0** , και αμέσως μετά ακολουθεί μια ***SUB*** που τον μειώνει κατά **1** ώστε να δείχνει στο τελευταίο γράμμα του αλφαριθμητικού. Επομένως στο σημείο αυτό, αν για παράδειγμα είχε δοθεί το string "RACECAR" θα βρισκόμασταν στην κατάσταση που φαίνεται στην εικόνα.



   Στη συνέχεια ακολουθεί το label **_PALINDROME_CHECK**, όπου ξεκινά και το loop μας και αρχικά συγκρίνονται οι τιμές των **R0** και **R3.** Όσο **R0 <= R3** τότε πρώτα φορτώνουμε στους **R1** και **R2** τις τιμές στις οποίες δείχνουν εκείνη τη

στιγμή οι **R0** και **R3**, ενώ αμέσως μετά αυξάνουμε τον **R0** και μειώνουμε τον **R3** κατά ένα, ώστε να δείχνουν στον επόμενο χαρακτήρα από τη πλευρά τους. Η παραπάνω διαδικασία αντιστοιχεί στην εντολή ***LDRB r1, [r0], #1*** και στην ***LDRB r2, [r3], #-1***. Ύστερα, συγκρίνουμε τους χαρακτήρες που έχουν φορτωθεί στους **R1,R2** και αν είναι **ίδιοι** τότε κάνουμε branch στην αρχή του loop, δηλαδή στο label **_PALINDROME_CHECK**. Ειδάλλως, το loop σταματά εκεί και ο κώδικάς συνεχίζει στο επόμενο block. Για παράδειγμα στο "RACECAR" , στο δεύτερο iteration, οι **R0,R3** θα δείχνουν στο πρώτο και τελευταίο «**A**» αντίστοιχα, άρα αυτοί οι χαρακτήρες θα φορτωθούν στους **R1,R2** , θα συγκριθούν και το loop θα συνεχιστεί. Αν είχαμε το string "**RACECER"** τότε ο **R1** θα περιείχε το «**A»** και ο **R2** το «**E**», άρα μετά τη σύγκριση το loop θα τελείωνε εκεί αφού ανιχνεύθηκε **μη** παλίνδρομο αλφαριθμητικό. Εάν όλοι οι χαρακτήρες είναι ίδιοι μέχρι τη στιγμή που **R0 > R3** τότε προφανώς δε χρειάζεται να συνεχίσουμε τις συγκρίσεις, καθώς από εκεί και μετά θα αρχίσουμε να συγκρίνουμε χαρακτήρες που έχουν ήδη συγκριθεί.

3. Τρίτο και τελευταίο block **_END**.
   Εδώ καταλήγει η ροή του κώδικα μετά το παραπάνω **loop.** Το αποτέλεσμα για το αν το string είναι παλίνδρομο ή όχι (**1** ή **0**) θα αποθηκευτεί στον **R0**. Αρχικοποιούμε τον **R0** με την τιμή **0** και ύστερα συγκρίνουμε τις τιμές που υπάρχουν τη στιγμή αυτή στους **R1,R2** δηλαδή τους χαρακτήρες που περιέχουν. Εδώ ξεχωρίζουμε **δύο** περιπτώσεις: **1.** Αν το αλφαριθμητικό ήταν παλίνδρομο τότε λόγω του παραπάνω **loop _PALINDROME_CHECK** όλοι οι χαρακτήρες θα έχουν συγκριθεί μεταξύ τους και οι **R1,R2** θα περιέχουν ίδιους χαρακτήρες. Επομένως απλά θα γίνει μία ακόμη σύγκριση. **2.** Εάν το αλφαριθμητικό **δεν** ήταν παλίνδρομο, τότε τη στιγμή που θα έγινε το branch στο **_END** οι **R1,R2** θα περιέχουν **διαφορετικούς** χαρακτήρες. Άρα με την χρήση της **conditional** εντολής ***MOVEQ r0, #1*** αμέσως μετά τη σύγκριση τοποθετούμε την τιμή **1** στον **R0 μόνο** αν έχουμε παλίνδρομο αλφαριθμητικό. Επιπλέον με τη χρήση της ψευδοεντολής ***MOV32 r3, 0x20000562*** φορτώνουμε στον **R3** μια συγκεκριμένη διεύθυνση και ύστερα με την ***STRB r0,[r3]*** αποθηκεύουμε στη διεύθυνση αυτή την τιμή του R0. Τέλος με τη ***BX lr*** κάνουμε return πίσω στη **main** όπου και καλέστηκε η συνάρτηση.

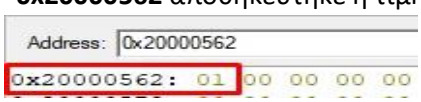
- ## Testing

  Για τη διαπίστωση της ορθής λειτουργίας του κώδικα δόθηκαν αρκετά διαφορετικά strings παλίνδρομα και μη, ενώ με τη βοήθεια του **debugger** παρατηρούνταν βήμα-βήμα πως φορτώνονται οι σωστές τιμές στους καταχωρητές. Ο κώδικας είναι case-sensitive και δε θεωρεί παλίνδρομο το string "Anna" αλλά θα έδινε θετικό αποτέλεσμα στο "anna" ή "ANNA". Επιπλέον δεν επιτρέπεται η ύπαρξη κενών, και αν υπάρχουν θα πρέπει η λέξη να έχει χωριστεί ακριβώς στη μέση. Ενδεικτικά μερικά strings που εξετάστηκαν:

  ✔. **"ANNA"** ✔. **"RACECAR"** ✔. **"RADAR"** ✔. **"NOON"**

  ✘. **"Anna"** ✘. **"NOTAPALINDROME"** ✘. **"RACE CAR"**

  Η τιμή επιστροφής αποθηκεύεται σε μία μεταβλητή τύπου **int** στη **main,** η οποία διαπιστώνουμε μέσω debugger ότι λαμβάνει τη **σωστή** τιμή (**0 ή 1**). Το ότι η διεύθυνση περιέχει τη σωστή τιμή μπορούμε να το δούμε και από το memory window του **Keil.** Και πράγματι όταν δόθηκε παλίνδρομο string, στη διεύθυνση **0x20000562** αποθηκεύτηκε η τιμή **1.**

- ## Προβλήματα

  Μοναδικό πρόβλημα υπήρξε η μη λειτουργία του **debugger** λόγω **read/write access violation.** Το πρόβλημα επιλύθηκε με το **initialization file** που ανέβηκε στο **elearning.**