Μικροεπεξεργαστές και Περιφερειακά

Εργασία 1η : Έλεγχος Παλίνδρομης Ακολουθίας

Authors

Χριστοφορίδης Σάββας AEM: 9147 email: schristofo@ece.auth.gr
Πορτοκαλίδης Σταύρος AEM: 9334 email: stavport@ece.auth.gr

Code:

Περιγραφή Προβλήματος

Σκοπός της πρώτης εργασίας είναι η υλοποίηση μιας ρουτίνας, σε assembly ARM, η οποία ελέγχει αν μια αλφαριθμητική ακολουθία είναι παλίνδρομη ή όχι. Εάν το αλφαριθμητικό είναι παλινδρομικό, τότε η ρουτίνα επιστρέφει την τιμή 1 σε μια θέση μνήμης, ενώ αν δεν είναι τότε επιστρέφει την τιμή 0 στην ίδια θέση μνήμης.

Αλγόριθμος

Η λογική του αλγορίθμου είναι απλή. Τοποθετούμε δύο δείκτες, έναν στον πρώτο και έναν στον τελευταίο χαρακτήρα του αλφαριθμητικού. Ελέγχουμε αν οι χαρακτήρες στις θέσεις που δείχνουν οι δείκτες είναι ίδιοι. Αν είναι ίδιοι, προχωράμε τον πρώτο δείκτη μια θέση, ενώ γυρνάμε τον δεύτερο δείκτη μια θέση πίσω. Η διαδικασία συνεχίζεται όσο ο πρώτος δείκτης δείχνει σε θέση που είναι πιο πίσω από την θέση που δείχνει ο δεύτερος δείκτης, σε περίπτωση που ο πρώτος ξεπεράσει τον δεύτερο ή δείχνουν στην ίδια θέση το αλφαριθμητικό είναι παλινδρομικό. Αν σε κάποιο έλεγχο οι χαρακτήρες δεν είναι ίδιοι, τότε το αλφαριθμητικό δεν είναι παλινδρομικό.

Υλοποίηση ρουτίνας

Η ***__asm void palindrome(const char* str, int* result)*** είναι η ρουτίνα που ελέγχει αν ένα αλφαριθμητικό είναι παλινδρομικό ή όχι. Παίρνει για ορίσματα το ***str***, το οποίο είναι ένας δείκτης που δείχνει στον πρώτο χαρακτήρα του αλφαριθμητικού, καθώς και το ***result*** το οποίο είναι ένας δείκτης που δείχνει στη θέση μνήμης όπου η ρουτίνα θα επιστρέψει την τιμή 0 ή 1. Η ρουτίνα επιστρέφει 1 αν το αλφαριθμητικό είναι παλινδρομικό και 0 αν το αλφαριθμητικό δεν είναι παλινδρομικό.

Αρχικά, η ρουτίνα εκτελεί το loop για την αρχικοποίηση των δεικτών (init_loop). Πιο συγκεκριμένα, προσπελαύνουμε τις θέσεις μνήμης ξεκινώντας από την θέση που δείχνει ο str μέχρι να βρούμε το χαρακτήρα '\0' που είναι και το τέλος του αλφαριθμητικού. Στη συνέχεια βάζουμε την διεύθυνση της μνήμης του πρώτου χαρακτήρα στον r0 και την διεύθυνση του τελευταίου χαρακτήρα στον r4.

Έπειτα, η ρουτίνα εκτελεί το palindrome_loop, στο οποίο ουσιαστικά εκτελείται ο αλγόριθμος. Υπάρχει, ένα έλεγχος που ελέγχει αν οι χαρακτήρες είναι ίδιοι και ένας έλεγχος που ελέγχει αν ο πρώτος δείκτης έχει προσπεράσει τον δεύτερο. Σε περίπτωση, που ο πρώτος δείκτης δείχνει σε μεγαλύτερη θέση από τον δεύτερο, δηλαδή το αλφαριθμητικό είναι παλινδρομικό, το Branch οδηγείται στο label palindrome_exit, όπου η ρουτίνα επιστρέφει την τιμή 1. Σε διαφορετική περίπτωση, όπου ο έλεγχος των χαρακτήρων βρίσκει ανόμοιους χαρακτήρες το αλφαριθμητικό δεν είναι παλινδρομικό και η ρουτίνα επιστρέφει 0.

Προβλήματα που αντιμετωπίστηκαν

Στην υλοποίηση της εργασίας δεν αντιμετωπίστηκαν ιδιαίτερα προβλήματα. Τα προβλήματα που υπήρχαν δεν αφορούσαν την υλοποίηση της ρουτίνας σε assembly ARM ούτε γενικά την υλοποίηση του κώδικα, αλλά ήταν διαδικαστικά. Αρχικά, είχαμε το πρόβλημα στο debug σε επίπεδο simulation, όπου μας εμφάνιζε το μήνυμα **'Read/write access violation error'**, το οποίο όπως απαντήθηκε από το επίσημο site του Keil αλλά και από τους διδάσκοντες, έχει να κάνει με τα limitations που υπάρχουν στην προσομοίωση του Keil σε επίπεδο simulation. Τέλος, στο περιβάλλον του μVision το 'Dynamic Syntax Checking' εμφανίζει errors στην δήλωση της ρουτίνας **' error: expected '(' after 'asm' '** και **' error: expected ';' after top-level asm block '.** Αλλά, όπως έχει απαντηθεί από το επίσημο support στο site του Keil έχει να κάνει με τον ARM compiler Version 6 που χρησιμοποίει 'Dynamic Syntax Checking'. Εφόσον εμείς χρησιμοποιούμε Version 5 για να κάνουμε Build το project το πρόβλημα αυτό το αγνοήθηκε.

Testing

Για τον έλεγχο ορθότητας της άσκησης βασιστήκαμε στην παρακάτω λογική.

Αφού πειστήκαμε ότι η λογική του αλγορίθμου είναι ορθή και μπορεί να αναγνωρίσει αν μια συμβολοσειρά είναι παλινδρομική ή όχι, τρέξαμε το project για ένα μεγάλο πλήθος παραδειγμάτων γραμμή-γραμμή με την βοήθεια του debugger με τρόπο ανάλογο που παρουσιάστηκε στο εργαστήριο του μαθήματος. Έτσι, με την χρήση των εργαλείων που προσφέρει το Keil, παρατηρήσαμε κάθε στιγμή τις τιμές των καταχωρητών και των μεταβλητών, καθώς και τις θέσεις που δείχνουν οι δείκτες. Για όλα τα παραδείγματα που δοκιμάσαμε οι τιμές ήταν οι αναμενόμενες και ο αλγόριθμος εκτελούνταν ορθά. Στη main του παραδοτέου υπάρχουν τρία παραδείγματα, **str** είναι η συμβολοσειρά που θα εξετάσουμε, το **result** δείχνει στην θέση μνήμης που έχει δεσμευτεί για την επιστροφή του αποτελέσματος και το **flag** είναι η μεταβλητή που μετά την εκτέλεση της ρουτίνας παίρνει την τιμή 1 αν η συμβολοσειρά είναι παλινδρομική και 0 αν δεν είναι, ανάλογα με την τιμή που επέστρεψε η ρουτίνα. Ουσιαστικά, μας βοηθάει να ελέγχουμε αν η ρουτίνα επέστρεψε την αναμενόμενη τιμή.