Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

Εργαστήριο Μικροϋπολογιστών

Τρίτη εργαστηριακή άσκηση

Σπουδαστές
Παπαλεξανδράκης Εμμανουήλ (Α.Μ.: 03114203)
Παπασκαρλάτος Αλέξανδρος (Α.Μ.: 03111097)

Ημερομηνία Υποβολής Αναφοράς: 4 Νοεμβρίου 2018

Θα παρουσιάσουμε μια σύντομη και ουσιαστική ανάλυση των προγραμμάτων που κατασκευάσαμε.
Τα προγράμματα αυτά καθ' αυτά είναι σε ξεχωριστά αρχεία.
Κάποιες λεπτομέρειες θα παραλειφθούν, αλλά εάν ο αναγνώστης ενδιαφέρεται, μπορεί να ανατρέξει στον κώδικα και στα σχόλια αυτού, όπου έχουμε μια πιο αναλυτική παρουσίαση.

Άσκηση 1

Κατασκευάζουμε το πρόγραμμα που ζητείται.

Το μόνο ενδιαφέρον σημείο είναι μάλλον η ρουτίνα χρονοκαθυστέρησης.
Κατασκευάζουμε μία δικιά μας τέτοια ρουτίνα.
Επειδή η πλακέτα λειτουργεί με συχνότητα 8MHz, χρειαζόμαστε ~4.000.000 κύκλους για να πετύχουμε χρονοκαθυστέρηση 0.5 sec.
Για να το πετύχουμε, δημιουργούμε ένα loop 80 κύκλων, το οποίο θα εκτελεστεί 50.000 φορές.
Στην πραγματικότητα, η ρουτίνα μας δημιουργεί χρονοκαθυστέρηση x*0.01 msec, όπου x θα είναι ένας αριθμός εώς 16 bit που θα δίνεται στο ζεύγος r25:r24 (εδώ x = 50.000).

Επειδή θα χρησιμοποιήσουμε ρουτίνες (στην πραγματικότητα μια ρουτίνα χρονοκαθυστέρησης), αρχικοποιούμε το δείκτη στοίβας sp. Θυμίζουμε πως με την κλήση της ρουτίνας αποθηκεύεται στη στοίβα η τιμή του pc, ώστε να μπορούμε να επιστρέψουμε στο ίδιο σημείο μετά το πέρας της ρουτίνας.
Έπειτα, ορίζουμε κατάλληλα τις θύρες για είσοδο - έξοδο δεδομένων.

Ανάβουμε το πρώτο led και μπαίνουμε στο loop αριστερής κίνησης (αφού πρώτα καλέσουμε τη ρουτίνα χρονοκαθυστέρησης 0.5 sec).
Σε κάθε iteration του loop αριστερής κίνησης, ελέγχουμε αν το pinB,0 είναι set, κι αν είναι, περιμένουμε μέχρι να γίνει clear.
Όταν θα γίνει clear, κάνουμε μία φορά left shift την έξοδό μας.
Καλούμε τη ρουτίνα χρονοκαθυστέρησης.
Ελέγχουμε αν φτάσαμε στο όριο της κίνησής μας. Αν φτάσαμε όντως στο MSB, περνάμε στη δεξιά κίνηση, διαφορετικά επαναλαμβάνουμε το loop αριστερής κίνησης.

Ακολουθούμε την ίδια ακριβώς διαδικασία για τη δεξιά κίνηση.
Οι μόνες διαφορές είναι προφανώς πως κάνουμε right shift και πως αν φτάσουμε στο LSB, περνάμε στην αριστερή κίνηση.

Άσκηση 2

Κατασκευάζουμε το πρόγραμμα που ζητείται.

Στον κώδικα της assembly, χρησιμοποιούμε συνολικά 5 καταχωρητές.
Έχουμε έναν να κρατάει ακέραιο το αρχικό input, δύο temp να διαχειρίζονται ενδιάμεσα αποτελέσματα και δύο ακόμα καταχωρητές που θα κρατούν τα αποτελέσματα για F0 και F1. Για το F2 θα χρησιμοποιήσουμε έναν απ' τους temp.

Θα παίξουμε μόνο με τα εκάστοτε lsb αγνοώντας τα υπόλοιπα bit. Βασικές εντολές είναι η lsr (για να περνάμε το bit που θέλουμε στο lsb), η mov (προφανώς) και οι λογικές εντολές and, or, com για τη σωστή υλοποίηση των συναρτήσεων Boole.

Χρησιμοποιώντας τους temp και τον καταχωρητή για F0, δημιουργούμε διαδοχικά τους όρους : AB, BC, AB+BC, CD, AB+BC+CD, DE, AB+BC+CD+DE, (AB+BC+CD+DE)' και αποθηκεύουμε την τελική τιμή στον F0.
Χρησιμοποιούμε μια μάσκα 0b 0000 0001, για να κρατήσουμε μόνο το lsb.

Στη συνέχεια, χρησιμοποιώντας τους temp και τον καταχωρητή για F1, δημιουργούμε διαδοχικά τους όρους: A, AB, ABC, ABCD, D', D'E', ABCD+D'E' και αποθηκεύουμε την τελική τιμή στον F1.
Χρησιμοποιούμε μια μάσκα 0b 0000 0001, για να κρατήσουμε μόνο το lsb.

Δημιουργουμε την F2 στον temp, ως F2 = F0 + F1.

Περνάμε τα αποτελέσματα όλων των συναρτήσεων στον temp, αφού κάνουμε κατάλληλα shift, έτσι ώστε η F2 να είναι στο bit 2 του temp, η F1 στο bit 1 και η F0 στο bit 0.
Βγάζουμε ως έξοδο το temp.

Όσον αφορά τον κώδικα στη C, η κύρια διαφορά είναι πως μπορούμε να εκμεταλλευτούμε τις δυνατότητες της C για να υλοποιήσουμε τις συναρτήσεις σε μία γραμμή κώδικα, εφόσον έχουμε αποθηκεύσει πρώτα κατάλληλα τα A, B, C, D, E (αποφασίσαμε να μην είμαστε τόσο φειδωλοί σε αυτήν την υλοποίηση).

Είναι F0 = ~((A&B) | (B&C) | (C&D) | (D&E))
F1 = (A & B & C & D) | (~D & ~E)
F2 = F0 | F1

Άσκηση 3

Κατασκευάζουμε το πρόγραμμα που ζητείται.

Καθ' υπόδειξη της εκφώνησης, όλες οι αλλαγές γίνονται αφήνοντας τα push buttons, συνεπώς, για να εκτελεστεί μια λειτουργία απαιτούμε το αντίστοιχο bit εισόδου να γίνει πρώτα 1 και μετά 0.

Για να το πετύχουμε αυτό στον κώδικά μας, έχουμε δύο μεταβλητές: input και state.
Η input θα έχει τα bits της εισόδου μας σε κάθε στιγμή.
Η state θα αρχικοποιείται στο 0 και θα γίνεται 1 ανν το αντίστοιχο bit εισόδου γίνει κάποια στιγμή 1. Σημειώνουμε πως, σε αντίθεση με την input, η state θα παραμένει στο 1 ακόμα κι αν η είσοδος επιστρέψει στο 0.

Ακολούθως, έχουμε ένα loop ελέγχου. Με σειρά από το bit 3 εώς το bit 0 (δηλαδή με τη σειρά προτεραιότητας που δίνεται), ελέγχουμε το bit της state και της input.
Ανν το bit της state είναι set (δηλαδή κάποια στιγμή η είσοδος έγινε 1), και το bit της input είναι clear (δηλαδή η είσοδος επέστρεψε στο 0), επιτελείται η αντίστοιχη λειτουργία.
Επιπλέον, μηδενίζουμε το bit στη state (άρα αρχικοποιούμε) της λειτουργίας που μόλις "ικανοποιήθηκε". Έπειτα, επιστρέφουμε στην αρχή και παίρνουμε νέα είσοδο.

Αν οι παραπάνω συνθήκες δεν είναι αληθείς για κανένα bit, τότε απλά επιστρέφουμε στην αρχή χωρίς καμία αλλαγή.