Εργαστήριο 3

Χειμερινό Εξάμηνο 2016-2017

Άσκηση 1- Αναδρομή (5 μονάδες)

Η ρωμαϊκή αρίθμηση χρησιμοποιεί τα παρακάτω ψηφία:

- M = 1000
- D = 500
- C = 100
- L = 50
- X = 10
- V = 5
- I = 1

Σε γενικές γραμμές, τα ρωμαϊκά ψηφία είναι γραμμένα σε φθίνουσα σειρά από αριστερά προς τα δεξιά, και προστίθενται διαδοχικά, για παράδειγμα MMXVI (=2016) ερμηνεύεται ως 1000 + 1000 + 10+5 + 1.
Ορισμένοι συνδυασμοί όμως τηρούν την αφαιρετική αρχή, η οποία ορίζει ότι όταν ένα σύμβολο μικρότερης αξίας προηγείται ενός συμβόλου μεγαλύτερης αξίας, η μικρότερη τιμή αφαιρείται από τη μεγαλύτερη τιμή, και το αποτέλεσμα προστίθεται στο σύνολο. Για παράδειγμα, MCMXLIV (=1944), τα σύμβολα C, X και I προηγούνται ένα σύμβολο μεγαλύτερης αξίας, και το αποτέλεσμα ερμηνεύεται ως εξής:
1000 + (1000 - 100) + (50 - 10) + (5 - 1).

1 = I	11 = XI	10 = X	100 = C	1 000 = M
2 = II	12 = XII	20 = XX	200 = CC	2 000 = MM
3 = III	13 = XIII	30 = XXX	300 = CCC	3 000 = MMM
4 = IV	14 = XIV	40 = XL	400 = CD	4 000 = MMMM
5 = V	15 = XV	50 = L	500 = D	
6 = VI	16 = XVI	60 = LX	600 = DC	
7 = VII	17 = XVII	70 = LXX	700 = DCC	
8 = VIII	18 = XVIII	80 = LXXX	800 = DCCC	
9 = IX	19 = XIX	90 = XC	900 = CM	

Να γραφτεί πρόγραμμα το οποίο θα διαβάζει από το πληκτρολόγιο μία συμβολοσειρά και θα καλέσει ανάλογα τις δύο παρακάτω αναδρομικές συναρτήσεις:

α) (1 μονάδα) Αναδρομική συνάρτηση int check_roman(char *s)
Η συνάρτηση αυτή επιστρέφει 1 όταν η συμβολοσειρά αποτελείται μόνο από ρωμαϊκά ψηφία, αλλιώς επιστρέφει 0. Μπορείτε να θεωρήσετε ότι δεχόμαστε μόνο τα κεφαλαία αντίστοιχα γράμματα: DCXI επιστρέφει 1, dCXi επιστρέφει 0, dcxi επιστρέφει 0.

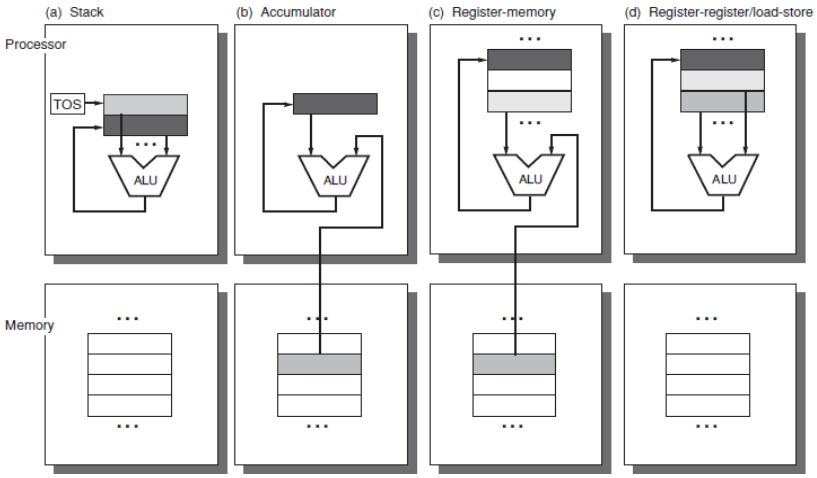
β) (4 μονάδες) Αναδρομική συνάρτηση int roman_to_decimal(char *s)
Η συνάρτηση αυτή δέχεται μία συμβολοσειρά που αποτελείται μόνο από ρωμαϊκά ψηφία (δηλαδή η main θα την καλέσει μόνο εφόσον έχει επιστρέψει 1 η συνάρτηση check_roman). Θα υπολογίσει τον αντίστοιχο δεκαδικό αριθμό και θα τον επιστρέψει. Στην συνέχεια η main θα εκτυπώσει στην οθόνη το αποτέλεσμα της roman_to_decimal εφόσον η check_roman επέστρεψε 1, αλλιώς θα εμφανίσει κατάλληλο μήνυμα λάθους στην οθόνη και θα διαβάσει νέα συμβολοσειρά από το πληκτρολόγιο μέχρι που να δοθεί έγκυρη συμβολοσειρά.

Μία διευκρίνηση για την άσκηση αναδρομής. Αναδρομή είναι ο ορισμός μίας συνάρτησης F χρησιμοποιώντας πάλι την ίδια συνάρτηση με διαφορετικές όμως παραμέτρους. Για παράδειγμα ο ορισμός του παραγοντικού ως F(n) = n*F(n-1), n>1 και F(0)=0 είναι μια αναδρομική σχέση, ενώ ο ορισμός F(n) = 1*2*…*(n-1)*n, δεν είναι αναδρομική σχέση. Συνεπώς, οι σωστές αναδρομικές λύσεις θα πρέπει να καλούν την ίδια συνάρτηση με διαφορετικές όμως τιμές στις παραμέτρους $a0, $a1 κοκ. για κάθε κλήση της F κατά την διάρκεια εκτέλεσης του προγράμματος. Λύσεις που δεν βασίζονται σε αυτήν την αρχή και που απλά κάνουν χρήση της εντολής κλήσης *jal F* σαν ένα απλό *goto* δεν θα γίνονται δεκτές.
Είναι σημαντικό η υλοποίηση των ασκήσεων να γίνει με κλήση συναρτήσεων ακριβώς όπως περιγράφει η εκφώνηση. Κάθε συνάρτηση πρέπει να εξασφαλίζει ότι οι λεγόμενοι Saved Temporary καταχωρητές διατηρούν το περιεχόμενο που είχαν πριν την κλήση της.
Μην κάνετε άσκοπα χρήση της στοίβας. Η χωρίς όρια δέσμευση περιττής μνήμης αποτελεί προγραμματιστικό λάθος.

Άσκηση 2. Απόδοση διαφόρων αρχιτεκτονικών (5 μονάδες)

Θέλουμε να αξιολογήσουμε τις τέσσερεις αρχιτεκτονικές που περιγράφονται στα γραφήματα 1 και 2 όσον αφορά την αποδοτικότητά τους να προσπελαύνουν την κύρια μνήμη. Για να μετρήσουμε την αποδοτικότητα αυτή κάνουμε τις εξής παραδοχές όσον αφορά τα τέσσερα σύνολα εντολών:

- Όλες οι εντολές έχουν μήκος πολλαπλάσιο του ενός byte
- Το opcode των εντολών έχει μήκος 1 byte
- Η προσπέλαση της μνήμης για δεδομένα γίνεται αποκλειστικά χρησιμοποιώντας absolute addressing (βλ. γράφημα 2).
- Οι μεταβλητές A, B, C και D είναι αρχικά στην μνήμη.



Γράφημα 1. Οι τέσσερεις βασικές αρχιτεκτονικές της άσκησης ανάλογα με την τοποθεσία των ορισμάτων (operands) εισόδου και εξόδου. Τα κουτιά με ανοιχτό γκρι χρώμα είναι είσοδοι στην ALU, ενώ τα κουτιά με σκούρο γκρι χρώμα είναι έξοδοι της ALU. Στην αρχιτεκτονική stack (a) η ALU παίρνει εισόδους αποκλειστικά από την στοίβα (stack) και συγκεκριμένα τα δύο ορίσματα που είναι στην κορυφή της στοίβας, το ένα κάτω από το άλλο. Ο pointer TOS (Top Of Stack) δείχνει κάθε χρονική στιγμή την κορυφή της στοίβας. Η ALU παράγει το αποτέλεσμα της πράξης το οποίο και τοποθετείται στην θέση του δεύτερου μετά την κορυφή ορίσματος ενώ το πρώτο όρισμα βγαίνει από την στοίβα και διαγράφεται. Στην αρχιτεκτονική Accumulator (b) οι πράξεις γίνονται μεταξύ του Accumulator και της μνήμης, και το αποτέλεσμα μεταβαίνει στον Accumulator (Ο Accumulator είναι απλά ένας καταχωρητής). Στην αρχιτεκτονική register-memory (c) (πχ Intel's x86), μπορούμε να εκτελέσουμε πράξεις απευθείας μεταξύ καταχωρητών και μνήμης, ενώ το αποτέλεσμα μπαίνει στον καταχωρητή. Η τελευταία αρχιτεκτονική register-register (d), είναι όμοια με την αρχιτεκτονική του MIPS.

Stack	**Accumulator**	**Register-Memory**	**Register-Register**
Push A # load from Mem[A] Push B # load from Mem[B] Add Pop C #write TOS to Mem[B]	Load A #Accum = A Add B Store C	Load R1, A Add R2, R1, B Store R2, C	Load R1, A Load R2, B Add R3, R1, R2 Store R3, C

Γράφημα 2. Ο κώδικας C=A+B για κάθε μία από τις τέσσερεις αρχιτεκτονικές του Γραφήματος 1. Προσέξτε ότι η εντολή Add στην αρχιτεκτονική stack δεν περιέχει εισόδους, επειδή οι είσοδοι υπονοείται ότι είναι οι δύο μεταβλητές στην κορυφή της λίστας. Οι μεταβλητές A, B και C βρίσκονται στην μνήμη και οι μεταβλητές A και B πρέπει να διατηρηθούν και μετά την εκτέλεση της εντολής C=A+B. Η εντολή *Pop A* βγάζει την μεταβλητή από την θέση TOS της στοίβας και την βάζει στην μνήμη.
Στην αρχιτεκτονική Accumulator θα πρέπει να αρχικοποιήσουμε τον Accumulator αρχικά με την εντολή Load για να μπορέσουμε να τον χρησιμοποιήσουμε.

a) Χρησιμοποιώντας εντολές assembly σαν αυτές που φαίνονται στο γράφημα 2, να αναπτύξετε τον παρακάτω κώδικα για κάθε μία από τις τέσσερεις αρχιτεκτονικές:

A = B+C;
B = A+C;
D = A-B;

[3]

Θα πρέπει ο κώδικάς σας να εκτελεί τον ελάχιστο αριθμό εντολών για κάθε αρχιτεκτονική. Θεωρείστε ότι μόνο η μεταβλητή D μας χρειάζεται μετά το τέλος των εντολών αυτών και συνεπώς θα πρέπει να αποθηκευθεί στην μνήμη *Mem*. Οι υπόλοιπες μεταβλητές A, B και C δεν χρειάζονται στον υπόλοιπο κώδικα μετά τις 3 αυτές εντολές.

b) Θεωρείστε τώρα ότι η παραπάνω ακολουθία των τριών εντολών εκτελείται σε επεξεργαστές με 16-bit διεύθυνση μνήμης και ότι οι μεταβλητές A, B, C και D έχουν μέγεθος επίσης 16 bits. Στην περίπτωση που χρειάζονται, υπάρχουν 16 καταχωρητές. Για κάθε μία αρχιτεκτονική ξεχωριστά και βασιζόμενοι στον κώδικα που αναπτύξατε στο ερώτημα (a), να απαντήσετε στα παρακάτω ερωτήματα:
   i. Πόσα bytes εντολών διαβάζονται από την μνήμη συνολικά;
   ii. Πόσα bytes δεδομένων διαβάζονται από και γράφονται στην μνήμη συνολικά;
   iii. Ποια αρχιτεκτονική είναι η πιο αποδοτική όσον αφορά την συνολική επικοινωνία με την μνήμη;

Θα πρέπει να στέλνετε με email τις λύσεις των εργαστηριακών ασκήσεων (την πρώτη άσκηση) σας στους διδάσκοντες στο uth.ece232lab@gmail.com.