Πανεπιστήμιο Θεσσαλίας
Τμήμα Πληροφορικής

Εισαγωγή στους Η/Υ

Δεύτερη Σειρά Ασκήσεων

8 Μαΐου 2019
παράδοση: *6μμ 22 Μαΐου*

Άσκηση 1:

Να αποδείξετε ότι κάθε πραγματικός αριθμός ο οποίος σε δυαδική αναπαράσταση σταθερής υποδιαστολής μπορεί να εκφραστεί με πεπερασμένο πλήθος κλασματικών ψηφίων, θα εκφράζεται με πεπερασμένο – πιθανά διαφορετικό – πλήθος κλασματικών ψηφίων και στην ισοδύναμη δεκαδική αναπαράσταση. Να δείξετε ότι το αντίστροφο δεν ισχύει.

Να εξετάσετε αν ισχύει το ίδιο μεταξύ τριαδικής και δεκαδικής αναπαράστασης, καθώς και μεταξύ πενταδικής και δεκαδικής αναπαράστασης.

Άσκηση 2:

Θεωρήστε μια αναπαράσταση προσημασμένων ακεραίων βάσης B, στην οποία τα ψηφία αναγράφονται σε φθίνουσα σειρά σημαντικότητας, με το λιγότερο σημαντικό ψηφίο να αντιστοιχεί σε θετική, και τα επόμενα ψηφία σε εναλλάξ αρνητική και θετική τιμή. Έτσι, ένας αριθμός αυτής της αναπαράστασης μεγέθους n ψηφίων:

$$A \equiv A_{n-1}A_{n-2}\ldots A_1A_0$$

έχει τιμή:

$$T(A) = (-1)^{n-1}\times T(A_{n-1})\times B^{n-1} + (-1)^{n-2}\times T(A_{n-2})\times B^{n-2} + \ldots - T(A_1)\times B + T(A_0)$$

με κάθε ψηφίο να έχει τιμή από 0 έως B-1.
Για παράδειγμα, αν B = 3, ο εξαψήφιος αριθμός 200112 της παραπάνω αναπαράστασης έχει τιμή $-2\times3^5 + 0\times3^4 - 0\times3^3 + 1\times3^2 - 1\times3 + 2 = -478$.

Α. Ποιο είναι το εύρος της παραπάνω αναπαράστασης για τυχαία B και n; Ποιος είναι ο αριθμός με τη μέγιστη και ποιος είναι ο αριθμός με την ελάχιστη τιμή;

Β. Δώστε έναν αλγόριθμο μετατροπής ενός τυχαίου αριθμού σταθερής υποδιαστολής του δεκαδικού συστήματος στην παραπάνω αναπαράσταση, και επαληθεύστε τον για τρεις τυχαίους αριθμούς.

Γ. Να εξετάσετε τη μοναδικότητα της αναπαράστασης· να εξετάσετε δηλαδή αν είναι δυνατό δύο διαφορετικοί αριθμοί της αναπαράστασης να έχουν ίδια τιμή. Να δώσετε αντιπαράδειγμα αν κάτι τέτοιο είναι δυνατό, ή διαφορετικά να αποδείξετε μαθηματικά τη μοναδικότητα. Αν βρήκατε ότι δεν υπάρχει μοναδικότητα στην αναπαράσταση ενός αριθμού, ποιο είναι το μέσο πλήθος διαφορετικών απεικονίσεων της ίδιας τιμής στην παραπάνω αναπαράσταση;

Δ. Δώστε έναν αλγόριθμο μετατροπής της παραπάνω αναπαράστασης στην αναπαράσταση συμπληρώματος ως προς B, και επαληθεύστε τον για τρεις τυχαίους αριθμούς, έναν για n=3, έναν για n=8 και έναν για n=13. Στη συνέχεια, και για B = 2, δώστε ένα συνδυαστικό κύκλωμα που με είσοδο έναν αριθμό της αρχικής αναπαράστασης δίνει τον αντίστοιχο αριθμό της νέας αναπαράστασης.

Ε. Επιστρέψτε στην αρχική αναπαράσταση και δώστε έναν αλγόριθμο πρόσθεσης/αφαίρεσης για αριθμούς της αναπαράστασης, συμπεριλαμβάνοντας έλεγχο υπερχείλισης, και επαληθεύστε τον για δύο τυχαία ζεύγη αριθμών, ένα για n=5 και ένα για n=11, και για τις δύο πράξεις ανά ζεύγος. Τέλος, και για B=2, δώστε ένα συνδυαστικό κύκλωμα που με είσοδο δύο αριθμούς

της αναπαράστασης και ένα σήμα επιλογής πρόσθεσης ή αφαίρεσης, εκτελεί την αντίστοιχη πράξη και δίνει ως έξοδο το αποτέλεσμά της καθώς και ένα σήμα υπερχείλισης.

Άσκηση 3:

Εξετάστε την αλήθεια των πιο κάτω προτάσεων σχετικά με μη προσημασμένους δυαδικούς ακέραιους, χρησιμοποιώντας αντιπαραδείγματα για να βρείτε τις ψευδείς και αποδεικνύοντας λογικά τις αληθείς προτάσεις:

Α. Όλοι οι μη μηδενικοί ακέραιοι που είναι πολλαπλάσια του 3 και έχουν ακριβώς δύο ψηφία '1' στην αναπαράστασή τους, έχουν περιττή διαφορά σημαντικότητας στα δύο αυτά ψηφία.

Β. Όλοι οι μη μηδενικοί ακέραιοι που είναι πολλαπλάσια του 6 έχουν *ακριβώς* δύο ψηφία '1' στην αναπαράστασή τους.

Γ. Όλοι οι μη μηδενικοί ακέραιοι που είναι πολλαπλάσια του 7 έχουν *τουλάχιστον* τρία ψηφία '1' στην αναπαράστασή τους.

Δ. Όλοι οι ακέραιοι που είναι πολλαπλάσια του 9 και έχουν *ακριβώς* τρία ψηφία '1' στην αναπαράστασή τους πρέπει να έχουν είτε περιττή είτε άρτια σημαντικότητα σε αυτά τα ψηφία.

Σημείωση: Σημαντικότητα είναι η δύναμη του 2 που αντιστοιχεί στο ψηφίο.

Άσκηση 4:

Ένας κώδικας Hamming προσφέρει κωδικοποίηση πληροφορίας με δυνατότητα ελέγχου και διόρθωσης λαθών. Ξεκινώντας από μια λέξη πληροφορίας των ν bits, δημιουργεί μια κωδικολέξη προσθέτοντας μ ψηφία ελέγχου ισοτιμίας, με τέτοιον τρόπο, ώστε να μπορεί να ανιχνεύσει διπλό και να διορθώσει απλό λάθος κατά την αποθήκευση ή τη διάδοση της κωδικοποιημένης πληροφορίας.
Αν η προς κωδικοποίηση λέξη είναι η $X \equiv X_1X_2\ldots X_\nu$ και τα ψηφία ελέγχου είναι τα $C_1, C_2, \ldots, C_\mu$, τότε η τελική κωδικολέξη είναι η $M \equiv M_1M_2\ldots M_{\nu+\mu}$ με τα ψηφία ελέγχου στις θέσεις 1,2, 4,8,…, δηλαδή $M \equiv C_1C_2X_1C_3X_2X_3X_4C_4X_5X_6\ldots$ Τα ψηφία αυτά κωδικοποιούν άρτια ισοτιμία με βάση τις σχέσεις:

$M_1 \oplus M_3 \oplus M_5 \oplus M_7 \oplus\ldots = 0$
$(M_2 \oplus M_3) \oplus (M_6 \oplus M_7) \oplus (M_{10} \oplus M_{11}) \oplus\ldots = 0$
$(M_4 \oplus M_5 \oplus M_6 \oplus M_7) \oplus (M_{12} \oplus M_{13} \oplus M_{14} \oplus M_{15}) \oplus\ldots = 0$
$(M_8 \oplus M_9 \oplus M_{10} \oplus M_{11} \oplus M_{12} \oplus M_{13} \oplus M_{14} \oplus M_{15}) \oplus\ldots = 0$
κοκ

Για τον έλεγχο ορθότητας της κωδικοποιημένης πληροφορίας κατά την ανάκτηση της κωδικολέξης από τη μνήμη ή το κανάλι διάδοσης, υπολογίζουμε τις τιμές ελέγχου ισοτιμίας C_1', C_2',…,C_μ' ως εξής:

$C_1' = M_1 \oplus M_3 \oplus M_5 \oplus M_7 \oplus\ldots$
$C_2' = (M_2 \oplus M_3) \oplus (M_6 \oplus M_7) \oplus (M_{10} \oplus M_{11}) \oplus\ldots$
$C_3' = (M_4 \oplus M_5 \oplus M_6 \oplus M_7) \oplus (M_{12} \oplus M_{13} \oplus M_{14} \oplus M_{15}) \oplus\ldots$
$C_4' = (M_8 \oplus M_9 \oplus M_{10} \oplus M_{11} \oplus M_{12} \oplus M_{13} \oplus M_{14} \oplus M_{15}) \oplus\ldots$
κοκ

Για άρτια ισοτιμία, η σωστή πληροφορία (ή τουλάχιστον αυτή που δεν έχει παραπάνω από δύο λάθη) επιβεβαιώνεται με τιμή 0 για όλες τις παραπάνω τιμές. Σε περίπτωση απλού λάθους, το ψηφίο λάθους εντοπίζεται στη θέση $C_\mu'\ldots C_2'C_1'$ και διορθώνεται.

Α. Για την παραπάνω κωδικοποίηση χρειαζόμαστε τόσα ψηφία ελέγχου, ώστε $2^\mu \geq \nu + \mu + 1$. Πόσα ψηφία ελέγχου χρειαζόμαστε για ν = 2, 4, 8, 16, 64;

Β. Να κωδικοποιηθεί κατά Hamming η λέξη πληροφορίας X = 1100110110 με άρτια ισοτιμία.

Γ. Εάν η κωδικολέξη που ανακτάται είναι η M = 00111001110110, να γίνει έλεγχος ορθότητας της πληροφορίας.

Δ. Να γίνει διόρθωση, υποθέτοντας απλό λάθος. Πήρατε την λέξη πληροφορίας του ερωτήματος Β;

Ε. Η παραπάνω μέθοδος έχει το μειονέκτημα ότι δε μπορεί να διαχωρίσει το απλό από το διπλό λάθος, και μπορεί έτσι να προχωρήσει σε λανθασμένη διόρθωση. Μια συνήθης διεύρυνση του κώδικα Hamming είναι η προσθήκη στο τέλος της κωδικολέξης και ενός ψηφίου συνολικής άρτιας ισοτιμίας, το οποίο χρησιμοποιείται μόνο για να διαχωρίσει το απλό από το διπλό λάθος και δε συμμετέχει στις παραπάνω σχέσεις ελέγχου ισοτιμίας. Αν λοιπόν το ψηφίο αυτό υποδείξει λάθος ισοτιμία, τότε είτε το λάθος βρίσκεται στο ίδιο το ψηφίο, οπότε ο πιο πάνω έλεγχος δεν θα δείξει λάθος, είτε υπάρχει αλλού ένα απλό λάθος που θα διορθωθεί με τον πιο πάνω έλεγχο. Αν όμως το ψηφίο αυτό υποδείξει σωστή ισοτιμία και ο πιο πάνω έλεγχος υποδείξει λάθος, τότε έχουμε διπλό λάθος που δεν επιδέχεται διόρθωση. Να επαναλάβετε τα ερωτήματα Β, Γ και Δ με το διευρυμένο κώδικα Hamming για την ίδια λέξη πληροφορίας Χ και με ανακτώμενη κωδικολέξη την M = 001110011101001. Τι παρατηρείτε;

Άσκηση 5:

Με βάση την εισαγωγική ανάλυση του κύκλου εντολής που κάναμε στο μάθημα, περιγράψτε τις φάσεις του κύκλου εντολής για τις ακόλουθες εντολές:

(α)	load ((Addr1),X)	για αρχιτεκτονική επέκτασης συσσωρευτή με έναν καταχωρητή-δείκτη X,
(β)	branch_less Addr2	για αρχιτεκτονική στοίβας,
(γ)	add $r3,104(Addr3,$r4*4)	για αρχιτεκτονική καταχωρητή-μνήμης,
(δ)	sub $r6,$r7,$r9	για αρχιτεκτονική φόρτωσης-αποθήκευσης, και
(ε)	move (Addr4,$r5),-4(Addr5,$r8)	για αρχιτεκτονική μνήμης-μνήμης

Για όλες τις περιπτώσεις υποθέστε ότι Addr1 έως Addr5 είναι διευθύνσεις μνήμης και ότι $r3 έως $r9 είναι καταχωρητές γενικού σκοπού. Οι παρενθέσεις υποδηλώνουν έμμεση διευθυνσιοδότηση είτε μέσω μνήμης είτε μέσω καταχωρητή, πιθανά συνδυασμένη με άλλες διευθυνσιοδοτήσεις. Οι μετατοπίσεις στα αριστερά παρενθέσεων προστίθενται στις διευθύνσεις που υπολογίζονται εντός των παρενθέσεων.

Προσέξτε ότι όπου υπάρχει αναφορά στη μνήμη, είτε σε ανάγνωση τελούμενων είτε σε αποθήκευση αποτελέσματος, της προσπέλασης μνήμης πρέπει να προηγείται η επίλυση της αναφοράς, δηλαδή ο υπολογισμός της τελικής διεύθυνσης προσπέλασης. Η επίλυση αναφοράς είναι πιθανό να περιλαμβάνει πρόσθετες προσπελάσεις μνήμης εάν υπάρχουν έμμεσες διευθυνσιοδοτήσεις μέσω μνήμης.

Άσκηση 6:

Θεωρήστε έναν επεξεργαστή που υλοποιεί μια παραλλαγή αρχιτεκτονικής στοίβας με το ακόλουθο σύνολο εντολών:

- Οι εντολές ADD, SUB, MUL και DIV για ακέραιες, και οι εντολές FADD, FSUB, FMUL και FDIV για πραγματικές αριθμητικές πράξεις, ορίζονται με υπονοούμενη διευθυνσιοδότηση στοίβας, όπου το δεξί τελούμενο εισόδου είναι το στοιχείο στην κορυφή της στοίβας, ενώ το αριστερό τελούμενο εισόδου είναι το στοιχείο αμέσως κάτω από την κορυφή της στοίβας. Τα δύο τελούμενα εισόδου αφαιρούνται από τη στοίβα, και στην κορυφή της στοίβας τοποθετείται το αποτέλεσμα.
- Η εντολή ADDI υλοποιεί ακέραια πρόσθεση με άμεσο τελούμενο, το οποίο είναι μια ακέραια σταθερά, με ή χωρίς πρόσημο. Στην περίπτωση αυτή, η πράξη γίνεται μεταξύ του αριθμού στην κορυφή της στοίβας και της σταθεράς αυτής. Μετά την εκτέλεση της πράξης, το στοιχείο στην κορυφή της στοίβας αντικαθίσταται με το αποτέλεσμα.

- Η εντολή FCONV υλοποιεί μετατροπή ακέραιου σε πραγματική αναπαράσταση. Η εντολή αυτή ορίζεται με υπονοούμενη διευθυνσιοδότηση, οπότε το στοιχείο στην κορυφή της στοίβας λαμβάνεται ως ακέραιος αριθμός, και αντικαθίσταται με τον ίδιο αριθμό, αλλά σε αναπαράσταση πραγματικού αριθμού.
- Η εντολή ZERO εισάγει στην κορυφή της στοίβας τη σταθερά 0 – ίδια για ακέραιους και πραγματικούς αριθμούς.
- Η εντολή DUP αντιγράφει το στοιχείο στην κορυφή της στοίβας, εισάγει δηλαδή στη στοίβα αντίγραφο της κορυφής της.
- Οι εντολές μεταφοράς δεδομένων μεταξύ μνήμης και στοίβας PUSH και POP ορίζονται με τον κλασικό τρόπο, δηλαδή με κατ' ευθείαν διευθυνσιοδότηση μνήμης, ενώ υποστηρίζεται και έμμεση διευθυνσιοδότηση μέσω μνήμης. Για παράδειγμα, η εντολή "PUSH (Δ)" εισάγει στη στοίβα μία λέξη από τη διεύθυνση μνήμης που *περιέχει* η διεύθυνση Δ.
- Διακλαδώσεις υλοποιούνται με τις εντολές BRZ και BRN, οι οποίες εκτελούν άλμα σε κάποια διεύθυνση κώδικα, αν το στοιχείο στην κορυφή της στοίβας έχει τιμή 0 ή <0, αντίστοιχα. Άμεσα άλματα υλοποιούνται με την εντολή JMP. Και στις τρεις αυτές εντολές η διεύθυνση προορισμού δίνεται ως μετατόπιση – συμβολικά ως ετικέτα, ενώ η στοίβα μένει ανέπαφη.
- Τέλος, η εντολή CLR αδειάζει τη στοίβα.

Αν η στοίβα έχει 16 θέσεις:

Α. Να γράψετε τον κώδικα στη συμβολική γλώσσα της πιο πάνω αρχιτεκτονικής για την αποτίμηση της αριθμητικής παράστασης πραγματικής τιμής:

$$S = \sum_{i=1}^{N} \{A_i \times \sum_{j=1}^{i-1} [\ |B_j - B_{j+1}| / j\ ]\}$$

Χρησιμοποιήστε τα A, B, S, I, J και N ως διευθύνσεις μνήμης, που περιέχουν: οι A και B τις *αρχικές διευθύνσεις* των ομώνυμων διανυσμάτων πραγματικών αριθμών, και οι S, I, J και N τις ομώνυμες βαθμωτές μεταβλητές, από τις οποίες η πρώτη είναι πραγματική και οι υπόλοιπες ακέραιες. Η μνήμη διευθυνσιοδοτείται σε λέξεις ίσου μεγέθους με το μέγεθος των δεδομένων. Πόσες εντολές εκτελούνται και πόσες φορές πρέπει να προσπελαστεί η μνήμη σε κάθε επανάληψη του βρόχου;

Β. Υποθέστε τώρα ότι στην παραπάνω αρχιτεκτονική προστίθενται οι νέες εντολές:

- Αριθμητικών πράξεων, οι ADDX, FADDX, SUBX, FSUBX, MULX, FMULX, DIVX και FDIVX, στις οποίες το αριστερό τελούμενο εισόδου καθορίζεται με σχετική διευθυνσιοδότηση από την κορυφή της στοίβας. Για παράδειγμα, η εντολή "SUBX 3" αφαιρεί το στοιχείο στην κορυφή της στοίβας από το στοιχείο που βρίσκεται 3 θέσεις πιο κάτω, ενώ η εντολή "MULX 0" πολλαπλασιάζει το στοιχείο στην κορυφή της στοίβας με τον εαυτό του. Με τις εντολές αυτές, από τη στοίβα αφαιρείται μόνο το στοιχείο στην κορυφή της, δηλαδή το δεξί τελούμενο εισόδου της πράξης, και στη θέση του τοποθετείται το αποτέλεσμα.

- Μεταφοράς δεδομένων, οι PUSHX και POPX, οι οποίες υλοποιούν έμμεση διευθυνσιοδότηση μνήμης με βάση κάποιο στοιχείο κάτω από την κορυφή της στοίβας. Για παράδειγμα, η εντολή "PUSHX 2" χρησιμοποιεί το στοιχείο που βρίσκεται σε απόσταση 2 από την κορυφή της στοίβας ως διεύθυνση μνήμης, από την οποία θα διαβάσει κάποια λέξη δεδομένων και θα τη φορτώσει στην κορυφή της στοίβας. Όπως και με τις προηγούμενες εντολές, μόνο η κορυφή της στοίβας επηρεάζεται από τις εντολές αυτές.

Να επαναλάβετε το ερώτημα Α, χρησιμοποιώντας όμως τις νέες εντολές της αρχιτεκτονικής, ώστε να ελαχιστοποιήσετε τον αριθμό προσπελάσεων μνήμης, διατηρώντας στη στοίβα τιμές που χρησιμοποιείτε συχνά.

Γ. Στη συνέχεια, θεωρήστε ότι η αρχιτεκτονική υποστηρίζει και τις εντολές INC και DEC, κατ' ευθείαν διευθυνσιοδότησης στη μνήμη, από τις οποίες η πρώτη αυξάνει και η δεύτερη μειώνει

το περιεχόμενο μιας θέσης μνήμης κατά 1 – χωρίς να αγγίζει τη στοίβα. Να επαναλάβετε το προηγούμενο ερώτημα, δεδομένων των νέων εντολών.

Δ. Τι παραπάνω θα θέλατε να έχετε στο σύνολο εντολών και στις μεθόδους διευθυνσιοδότησης της πιο πάνω αρχιτεκτονικής, ώστε ο κώδικας που γράψατε να γίνει πιο σύντομος και να επιτυγχάνει τον ελάχιστο δυνατό αριθμό προσπελάσεων μνήμης ανά επανάληψη;

Άσκηση 7:

Θεωρήστε έναν υπολογιστή αρχιτεκτονικής συσσωρευτή. Το σύνολο εντολών ακεραίων του επεξεργαστή αυτού περιλαμβάνει τις εντολές load, store, multiply, divide, add και sub, οι οποίες έχουν ένα τελούμενο Δ (κατ' ευθείαν διευθυνσιοδότηση μνήμης) και αναφέρονται στο συσσωρευτή Σ ως εξής:

Εντολή	Επεξήγηση
load Δ	Σ = MEM[Δ]
store Δ	MEM[Δ] = Σ
multiply Δ	Σ = Σ * MEM[Δ]
divide Δ	Σ = Σ / MEM[Δ]
add Δ	Σ = Σ + MEM[Δ]
sub Δ	Σ = Σ - MEM[Δ]

Οι παραπάνω εντολές – πλην της store – μπορούν να δεχτούν και άμεσο τελούμενο, το οποίο εκφράζεται με το σύμβολο '#' πριν το άμεσο τελούμενο. Για τα ερωτήματα που ακολουθούν υποθέστε ότι οι μεταβλητές και τα στοιχεία διανυσμάτων που θα μας απασχολήσουν είναι μεγέθους *λέξης*, και ότι η διευθυνσιοδότηση της μνήμης γίνεται σε *λέξεις*.

Α. Γράψτε ένα πρόγραμμα στη συμβολική γλώσσα της αρχιτεκτονικής αυτής, που να υπολογίζει την τιμή της ακέραιας παράστασης:

$$P = \prod_{i=0}^{n-1} \{ [A_i \times \sum_{j=0}^{i-1} B_j - A_{n-1-i} \times \sum_{j=i}^{n-1} B_j] / i! \}$$

για n = 4. Θεωρήστε τα A_i, B_i, για i=1,…,n, και P ως τις διευθύνσεις μνήμης των αντίστοιχων μεταβλητών. Τι παρατηρείτε στη μορφή του προγράμματος που γράψατε; Μπορείτε να χρησιμοποιήσετε το ίδιο πρόγραμμα για άλλη τιμή του n;

Β. Έστω ότι το σύνολο εντολών του επεξεργαστή επεκτείνεται, ώστε να περιλαμβάνει και τις εντολές inc, dec, cmp, beq και bne, με την ακόλουθη λειτουργία:

Εντολή	Επεξήγηση
inc Δ	MEM[Δ]++
dec Δ	MEM[Δ]--
cmp Δ	F = (Σ == MEM[Δ])
beq E	if (F) branch E
bne E	if (!F) branch E

όπου F κατάλληλο ψηφίο ελέγχου του επεξεργαστή, και E κάποια ετικέτα στον κώδικα που εκφράζει μετατόπιση σχετικά με την τιμή του μετρητή προγράμματος. Η εντολή cmp μπορεί να δεχτεί και άμεσο τελούμενο, με τον τρόπο που αναφέρθηκε νωρίτερα. Ο επεξεργαστής μπορεί τώρα να δεχτεί και έμμεση διευθυνσιοδότηση μνήμης για τις εντολές του που έχουν τελούμενο Δ, η οποία συμβολίζεται και περιγράφεται ως εξής:

Εντολή	Επεξήγηση
load (Δ)	Σ = MEM[MEM[Δ]]
store (Δ)	MEM[MEM[Δ]] = Σ

multiply (Δ)	Σ = Σ * MEM[MEM[Δ]]
divide (Δ)	Σ = Σ / MEM[MEM[Δ]]
add (Δ)	Σ = Σ + MEM[MEM[Δ]]
sub (Δ)	Σ = Σ - MEM[MEM[Δ]]
inc (Δ)	MEM[MEM[Δ]]++
dec (Δ)	MEM[MEM[Δ]]--
cmp (Δ)	F = (Σ == MEM[MEM[Δ]])

Γράψτε το πρόγραμμα που υπολογίζει την πιο πάνω παράσταση για μεταβλητό n, η τιμή του οποίου βρίσκεται στη διεύθυνση μνήμης N. Υποθέστε ότι οι διευθύνσεις μνήμης A και B περιέχουν τις *αρχικές διευθύνσεις* των αντίστοιχων διανυσμάτων ακεραίων, και ότι διαδοχικά στοιχεία των διανυσμάτων βρίσκονται σε διαδοχικές θέσεις μνήμης.

Γ. Επεκτείνετε την αρχιτεκτονική συσσωρευτή, ώστε να περιλαμβάνει έναν καταχωρητή δείκτη X. Οι παραπάνω εντολές μπορούν έτσι να δεχτούν δεικτοδοτούμενη διευθυνσιοδότηση, που συμβολίζεται και περιγράφεται όπως φαίνεται στον ακόλουθο πίνακα:

Εντολή	**Επεξήγηση**
load Δ,X	Σ = MEM[Δ+X]
store Δ,X	MEM[Δ+X] = Σ
multiply Δ,X	Σ = Σ * MEM[Δ+X]
divide Δ,X	Σ = Σ / MEM[Δ+X]
add Δ,X	Σ = Σ + MEM[Δ+X]
sub Δ,X	Σ = Σ - MEM[Δ+X]
inc Δ,X	MEM[Δ+X] = MEM[Δ+X] + 1
dec Δ,X	MEM[Δ+X]--
cmp Δ,X	F = (Σ == MEM[Δ+X])

όπου στη διεύθυνση Δ προστίθεται το περιεχόμενο του καταχωρητή X για τον υπολογισμό της τελικής διεύθυνσης προσπέλασης. Όλες οι εντολές του πίνακα μπορούν να δεχτούν *και* έμμεση διευθυνσιοδότηση, που εφαρμόζεται *πριν* τη δεικτοδοτούμενη.
Το σύνολο εντολών του επεξεργαστή περιλαμβάνει τώρα και τις πιο κάτω εντολές:

Εντολή	**Επεξήγηση**
loadX Δ	X = MEM[Δ]
storeX Δ	MEM[Δ] = X
cmpX Δ	F = (X == MEM[Δ])
incX	X++
decX	X--
bnzX E	if (X != 0) branch E
bezX E	if (X == 0) branch E

Οι εντολές loadX και cmpX δέχονται και άμεση διευθυνσιοδότηση, ενώ οι εντολές loadX, storeX και cmpX δέχονται *και* έμμεση διευθυνσιοδότηση μνήμης.
Ξαναγράψτε το πρόγραμμα που υπολογίζει την παραπάνω παράσταση για μεταβλητό n, χρησιμοποιώντας δεικτοδοτούμενη διευθυνσιοδότηση μέσω του καταχωρητή X. Όπως και πριν, υποθέστε ότι οι διευθύνσεις μνήμης A και B περιέχουν τις *αρχικές διευθύνσεις* των αντίστοιχων διανυσμάτων, και ότι διαδοχικά στοιχεία των διανυσμάτων βρίσκονται σε διαδοχικές θέσεις μνήμης.

Δ. Να συγκρίνετε τα προγράμματα που γράψατε στα δύο προηγούμενα ερωτήματα, υπολογίζοντας το συνολικό αριθμό εντολών που εκτελούνται, καθώς και το συνολικό αριθμό προσπελάσεων μνήμης που γίνονται. Τι συμπεράσματα βγάζετε για τη χρήση καταχωρητή δείκτη σε μια αρχιτεκτονική συσσωρευτή;

Άσκηση 8:

Θεωρήστε την αρχιτεκτονική στοίβας της εικονικής μηχανής java. Μέρος από τη συμβολική γλώσσα αυτής της αρχιτεκτονικής δίνεται στον Πίνακα 1. Με βάση αυτόν τον πίνακα, γράψτε τον κώδικα μιας συνάρτηση FUNC που υπολογίζει και επιστρέφει την τιμή της ακέραιας παράστασης της προηγούμενης άσκησης, για τυχαία τιμή του n, με την υπόθεση ότι η τιμή του n περνάει ως παράμετρος στην τοπική μεταβλητή 0, οι αρχικές διευθύνσεις των διανυσμάτων A και B περνούν επίσης ως παράμετροι στις τοπικές μεταβλητές 1 και 2, ενώ το αποτέλεσμα P θέλουμε στο τέλος να επιστραφεί στην τοπική μεταβλητή 0.
Προσπαθήστε για το μικρότερο δυνατό κώδικα σε bytes, χρησιμοποιώντας όσο μπορείτε τις τοπικές μεταβλητές 0, 1, 2 και 3 κατά τους υπολογισμούς σας. Υποθέστε ότι όλες οι θέσεις τοπικών μεταβλητών 4 – 255 είναι διαθέσιμες για φορτώσεις και αποθηκεύσεις προσωρινών τιμών. Να υπολογίσετε το μέγεθος σε bytes του κώδικα που γράψατε.

Υπενθύμιση: Οι ασκήσεις παραδίνονται μόνο χειρόγραφες.

Πίνακας 1. Επιλογή εντολών java bytecode

Εντολή	Πρόσθετα bytes	Στοίβα [πριν]→[μετά]	Περιγραφή
aload	1: index	→ objectref	φόρτωση διεύθυνσης από την τοπική μεταβλητή *#index*
aload_*n*		→ objectref	φόρτωση διεύθυνσης από την τοπική μεταβλητή *n*, όπου *n* = 0,1,2 ή 3
bipush	1: immbyte	→ value	φόρτωση σταθεράς *immbyte*
dup		value → value, value	αντιγραφή της κορυφής στοίβας
goto	2: targetshort		άλμα σε μετατόπιση 16-bit *targetshort*
iadd		value1, value2 → result	πρόσθεση ακεραίων
iaload		arrayref, index → value	φόρτωση ακεραίου από πίνακα
iastore		arrayref, index, value →	αποθήκευση ακεραίου σε πίνακα
iconst_m1		→ -1	φόρτωση της ακέραιας σταθεράς -1
iconst_*n*		→ 0	φόρτωση της ακέραιας σταθεράς *n*, όπου *n* = 0,1,2,3,4 ή 5
idiv		value1, value2 → result	διαίρεση ακεραίων
if_icmpX	2: branchshort	value1, value2 →	αν *value X value2*, άλμα σε μετατόπιση 16-bit *branchshort*, όπου X = eq, ge, gt, le, lt ή ne, για αντίστοιχη σύγκριση
ifnotnull	2: branchshort	value → value	αν *value != null*, άλμα σε μετατόπιση 16-bit *branchshort*
ifnull	2: branchshort	value → value	αν *value == null*, άλμα σε μετατόπιση 16-bit *branchshort*
iinc	2: index, const		αύξηση της τοπικής μεταβλητής *#index* κατά *const*
iload	1: index	→ value	φόρτωση ακεραίου από την τοπική μεταβλητή *#index*
iload_*n*		→ value	φόρτωση ακεραίου από την τοπική μεταβλητή *n*, όπου *n* = 0,1,2 ή 3
imul		value1, value2 → result	πολλαπλασιασμός ακεραίων
ineg		value → result	αντίθετος ακέραιος
irem		value1, value2 → result	υπόλοιπο διαίρεσης ακεραίων
ishl		value1, value2 → result	αριστερή ολίσθηση
ishr		value1, value2 → result	δεξιά αριθμητική ολίσθηση
istore	1: index	value →	αποθήκευση στην τοπική μεταβλητή *#index*

istore_*n*		value →	αποθήκευση στην τοπική μεταβλητή *n*, όπου *n* = 0,1,2 ή 3
isub		value1, value2 → result	αφαίρεση ακεραίων
iushr		value1, value2 → result	δεξιά λογική ολίσθηση
jsr	2: targetshort	→ address	άλμα σε μετατόπιση 16-bit *targetshort* και τοποθέτηση στη στοίβα της διεύθυνσης επανόδου
pop		value →	αφαίρεση κορυφής στοίβας
ret	1: index		άλμα στη διεύθυνση που περιέχεται στην τοπική μεταβλητή *#index*
sipush	2: immshort	→ value	φόρτωση σταθεράς 16-bit *immshort*
swap		value2, value1 → value1, value2	εναλλαγή των δύο στοιχείων στην κορυφή της στοίβας

Παρατηρήσεις:

1. Οι εντολές java bytecode καταλαμβάνουν 1 byte, εκτός αν η δεύτερη στήλη αναγράφει πρόσθετα bytes.
2. Αν η εντολή περιέχει 2 πρόσθετα bytes που αποτελούν αριθμό 16-bit (short), τότε τα bytes αυτά αναγράφονται στη συμβολική μορφή ως *ένας* αριθμός (εντολές *if_icmpX*, *sipush*, *goto*, *jsr*).
3. Το μοντέλο μνήμης παρέχει γρήγορη κατ' ευθείαν αναφορά σε 4 θέσεις (θέσεις 0, 1, 2 και 3) με εντολές φόρτωσης και αποθήκευσης μεγέθους 1 byte (εντολές *aload*_n, *iload*_n, *istore*_n, όπου το n είναι 0, 1, 2 ή 3).
4. Για αναφορά στη μνήμη πέρα από τις 4 παραπάνω θέσεις, το μοντέλο παρέχει τη δυνατότητα κατ' ευθείαν αναφοράς στις επόμενες 252 θέσεις με 1 πρόσθετο byte διεύθυνσης (index) (εντολές *aload*, *iload*, *istore*, *ret*).
5. Για ευρύτερη αναφορά στη μνήμη, παρέχεται έμμεση και δεικτοδοτούμενη διευθυνσιοδότηση μέσω στοίβας, οπότε η διεύθυνση μνήμης πρέπει να φορτωθεί στη στοίβα μαζί με έναν αριθμοδείκτη (μηδενικός για απλή έμμεση αναφορά) για φόρτωση από τη μνήμη, και μαζί με μια ακόμα τιμή για αποθήκευση στη μνήμη (εντολές *iaload*, *iastore*).
6. Η φόρτωση σταθερών τιμών στη στοίβα μπορεί να γίνει με εντολή μεγέθους 1 byte όταν πρόκειται για τις συγκεκριμένες συνήθεις τιμές -1,0,1,2,3,4,5 (εντολές *iconst_m1*, *iconst*_n, όπου το n είναι 0, 1, 2, 3, 4 ή 5), με 1 πρόσθετο byte για σταθερές 8-bit (εντολή *bipush*) ή με 2 πρόσθετα bytes για σταθερές 16-bit (εντολή *sipush*).
7. Οι εντολές διακλάδωσης με σύγκριση αφαιρούν τα δύο τελούμενα σύγκρισης από τη στοίβα (εντολές *if_icmpX*), ενώ οι εντολές διακλάδωσης με έλεγχο μηδενικής τιμής *δεν* αφαιρούν το τελούμενο ελέγχου από τη στοίβα (εντολές *ifnotnull*, *ifnull*).
8. Σε όλες τις εντολές διακλάδωσης και άλματος η μετατόπιση μπορεί συμβολικά να αναγραφεί είτε ως ένας αριθμός 16-bit είτε ως ετικέτα προορισμού (εντολές *if_icmpX*, *ifnotnull*, *ifnull*, *goto*, *jsr*).