ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

1η ΑΣΚΗΣΗ ΣΤΗΝ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ
Ακ. έτος 2010-2011, 5ο Εξάμηνο Σχολή ΗΜ&ΜΥ

ΜΕΡΟΣ Α

A1. Εκτελέστε την πράξη $57_{<10>}$ * $-45_{<10>}$ χρησιμοποιώντας τον αλγόριθμο Booth. Δείξτε βήμα-βήμα την εκτέλεση του αλγορίθμου. Πόσες πράξεις (προσθέσεις/αφαιρέσεις) απαιτούνται συνολικά; Υποθέστε ότι το σύστημα αναπαριστά τους ακεραίους με 7 bits σε συμπλήρωμα ως προς 2.

Ο αριθμός $57_{<10>}$ αντιστοιχεί στον 0111001 και ο $-45_{<10>}$ στον 1010011. Ο αντίθετος του 57 που θα χρησιμοποιηθεί όταν απαιτείται αφαίρεση είναι ο 1000111 (σε συμπλήρωμα ως προς 2). Η εκτέλεση του αλγορίθμου Booth φαίνεται παρακάτω :

Operation	A	Q	Q_{n+1}	n
	0000000	1010011	0	111
10 → sub	1000111			
	1000111	1010011		
ashr	1100011	1101001	1	110
11 → ashr	1110001	1110100	1	101
01 → add	0111001			
	0101010	1110100		
ashr	0010101	0111010	0	100
00 → ashr	0001010	1011101	0	011
10 → sub	1000111			
	1010001	1011101		
ashr	1101000	1101110	1	010
01 → add	0111001			
	0100001	1101110		
ashr	0010000	1110111	0	001
10 → sub	1000111			
	1010111	1110111		
ashr	1101011	1111011	1	000

Επομένως το τελικό αποτέλεσμα είναι ο αριθμός 11010111111011 = $-2565_{<10>}$. Για την παραγωγή του αποτελέσματος απαιτήθηκαν 5 πράξεις.

A2. Ως γνωστόν για τον πολλαπλασιασμό ισχύει AxB = BxA. Πως επηρεάζει όμως η αλλαγή αυτή την απόδοση του πολλαπλασιασμού στην περίπτωση του αλγορίθμου Booth; Θα άλλαζε κάτι στο προηγούμενο ερώτημα αν στηριζόμασταν στην αντιμεταθετική ιδιότητα και πραγματοποιούσαμε την πράξη $-45_{<10>}$ * $57_{<10>}$;

Ο αριθμός των πράξεων που εκτελούνται κατά τον αλγόριθμο Booth εξαρτάται από τις μεταβάσεις από bit 0 σε bit 1 και αντίστροφα που παρατηρούνται στον πολλαπλασιαστή. Όσο λιγότερες οι

μεταβάσεις τόσο λιγότερες και οι πράξεις. Στο συγκεκριμένο παράδειγμα, ο 57=0111001 παρουσιάζει 4 τέτοιες μεταβάσεις ενώ ο -45=1010011 παρουσιάζει 5 μεταβάσεις. Έτσι, ο πολλαπλασιασμός 57*(-45) απαιτεί περισσότερες πράξεις από τον (-45)*57.

Εκτελούμε τον ίδιο πολλαπλασιασμό, έχοντας πρώτα εφαρμόσει την αντιμεταθετική ιδιότητα.

Operation	A	Q	Q_{n+1}	n
	0000000	0111001	0	111
10 → sub	0101101			
	0101101	0111001		
ashr	0010110	1011100	1	110
01 → add	1010011			
	1101001	1011100		
ashr	1110100	1101110	0	101
00 → ashr	1111010	0110111	0	100
10 → sub	0101101			
	0100111	0110111		
ashr	0010011	1011011	1	011
11 → ashr	0001001	1101101	1	010
11 → ashr	0000100	1110110	1	001
01 → add	1010011			
	1010111	1110110		
ashr	1101011	1111011	0	000

Επομένως καταλήγουμε και πάλι στο ίδιο αποτέλεσμα, αλλά αυτή τη φορά έχοντας εκτελέσει 4 πράξεις αντί για 5.

ΜΕΡΟΣ Β

Β1. Υλοποιήστε τη διαδικασία (procedure) *pfind* σε assembly MIPS. Η διαδικασία αυτή δέχεται ως όρισμα στον καταχωρητή *$a0* ένα δείκτη στο πρώτο χαρακτήρα ενός *null-terminated string* και επιστρέφει στον καταχωρητή *$v0* τη διεύθυνση του πρώτου χαρακτήρα *p* του string. Αν το string δεν περιέχει τον χαρακτήρα p, τότε η pfind επιστρέφει τη διεύθυνση του χαρακτήρα null που βρίσκεται στο τέλος του string. Αν για παράδειγμα ο $a0 δείχνει στο string "mips", τότε το αποτέλεσμα που επιστρέφει θα πρέπει να είναι ένας δείκτης στον τρίτο χαρακτήρα του string.

```
pfind:
      add  $t0, $zero, "p"     # αποθήκευση στον $t0 του χαρακτήρα p, δηλαδή του
                               # 112<10>=70<16>
loop: lbu  $t1, 0($a0)         # φόρτωση στον $t1 του χαρακτήρα προς εξέταση
      addi $a0, $a0, 1         # μετάβαση στον επόμενο byte δηλαδή στον επόμενο
                               # χαρακτήρα
      beq  $t1, $t0, done      # σύγκριση με τον χαρακτήρα p
      bneq $t1, $zero, loop    # επανάληψη του loop μέχρι να βρεθεί είτε το p είτε
                               # ο χαρακτήρα NULL που δηλώνει το τέλος του string
done: addi $v0, $a0, -1        # αποθήκευση του σωστού δείκτη στο $v0
      jr   $ra
```

Β2. Υλοποιήστε τη διαδικασία (procedure) *pcount* σε assembly MIPS. Η διαδικασία αυτή δέχεται σαν όρισμα ένα δείκτη στον πρώτο χαρακτήρα ενός null-terminated string και επιστρέφει τη συχνότητα εμφάνισης του χαρακτήρα p στο string αυτό. Χρησιμοποιήστε τη διαδικασία pfind που υλοποιήσατε στο Β1.

```
pcount:
      addi $sp, $sp, -8      # αποθήκευση του $ra και του $s0 στην στοίβα αφού
      sw   $ra, 4($sp)       # σύμφωνα με το πρότυπο πρέπει η κλήση συναρτήσεων
      sw   $s0, 0($sp)       # να διατηρεί την τιμή στους καταχωρητές $s ενώ και
                             # ο $ra θα επηρεαστεί από το jal στη pfind
      add  $s0, $zero, $zero # μηδενισμός του counter
loop: jal  pfind
      addi $s0, $s0, 1       # counter++
      addi $a0, $v0, 1       # μετάβαση στον επόμενο χαρακτήρα του string
      lbu  $t1, 0($v0)
      bneq $t1, $zero, loop  # loop αν η pfind δεν έχει επιστρέψει δείκτη σε NULL
      addi $v0, $s0, -1      # επιστροφή του σωστού counter
      lw   $s0, 0($sp)       # αποκατάσταση των τιμών των $s0, $ra
      lw   $ra, 4($sp)
      addi $sp, $sp, 8
      jr   $ra
```

B3. Ο διωνυμικός συντελεστής $C(n,k) = \binom{n}{k}$ μπορεί να υπολογιστεί αναδρομικά με βάση την εξίσωση:

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k), & k > 0 \text{ και } k < n \\ 1, & k = 0 \text{ ή } k = n \\ 0, & k > n \end{cases}$$

όπου n, k μη αρνητικοί ακέραιοι. Υλοποιήστε σε assembly MIPS τη διαδικασία *binomcoeff*, η οποία θα δέχεται σαν όρισματα τους μη αρνητικούς ακεραίους n και k και θα επιστρέφει σαν αποτέλεσμα τον διωνυμικό συντελεστή $C(n,k)$.

```
binomcoeff:
      addi $sp, $sp, -16
      sw   $ra, 12($sp)
      sw   $s0, 8($sp)
      sw   $a1, 4($sp)
      sw   $a0, 0($sp)
      slti $t1, $a0, $a1      # έλεγχος αν n < k
      beq  $t1, $zero, L1
      addi $v0, $zero, $zero  # return 0
      addi $sp, $sp, 16
      jr   $ra
L1:   beq  $a1, $zero, L2     # έλεγχος αν k = 0
      beq  $a0, $a1, L2       # έλεγχος αν k = n
      j    recur
L2:   addi $v0, $zero, 1      # return 1
      addi $sp, $sp, 16
      jr   $ra
recur:
      addi $a0, $a0, -1
      addi $a1, $a1, -1
      jal  binomcoeff         # υπολογισμός του C(n-1, k-1)
      add  $s0, $v0, $zero
      lw   $a0, 0($sp)
      lw   $a1, 4($sp)
      addi $a0, $a0, -1
      jal  binomcoeff         # υπολογισμός του C(n-1, k)
      add  $v0, $v0, $s0
      lw   $ra, 12($sp)
      addi $sp, $sp, 16
      jr   $ra
```